SPRINGER
REFERENCE

Claude Sammut
Geoffrey I. Webb
Editors

Encyclopedia of Machine Learning and Data Mining

Second Edition

Springer

Encyclopedia of Machine Learning and Data Mining

Claude Sammut • Geoffrey I. Webb
Editors

Encyclopedia of Machine Learning and Data Mining

Second Edition

With 263 Figures and 34 Tables

Springer

Editors
Claude Sammut
The University of New South Wales
Sydney, NSW
Australia

Geoffrey I. Webb
Faculty of Information Technology
Monash University
Melbourne, VIC, Australia

ISBN 978-1-4899-7685-7 ISBN 978-1-4899-7687-1 (eBook)
ISBN 978-1-4899-7686-4 (print and electronic bundle)
DOI 10.1007/978-1-4899-7687-1

Library of Congress Control Number: 2016958560

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer Science+Business Media LLC
The registered company address is: 233 Spring Street, New York, NY 10013, U.S.A.

Preface

Machine learning and data mining are rapidly developing fields. Following the success of the first edition of the *Encyclopedia of Machine Learning*, we are delighted to bring you this updated and expanded edition. We have expanded the scope, as reflected in the revised title *Encyclopedia of Machine Learning and Data Mining*, to encompass more of the broader activity that surrounds the machine learning process. This includes new articles in such diverse areas as anomaly detection, online controlled experiments, and record linkage as well as substantial expansion of existing entries such as data preparation. We have also included new entries on key recent developments in core machine learning, such as deep learning. A thorough review has also led to updating of much of the existing content.

This substantial tome is the product of an intense effort by many individuals. We thank the Editorial Board and the numerous contributors who have provided the content. We are grateful to the Springer team of Andrew Spencer, Michael Hermann, and Melissa Fearon who have shepherded us through the long process of bringing this second edition to print. We are also grateful to the production staff who have turned the content into its final form.

We are confident that this revised encyclopedia will consolidate the first edition's place as a key reference source for the machine learning and data mining communities.

Contributors

Pieter Abbeel EECS Department, UC Berkeley, Stanford, CA, USA

Zahraa S. Abdallah Faculty of Information Technology, Monash University, Clayton, VIC, Australia

Charu C. Aggarwal IBM T. J. Watson Research Center, Hawthorne, NY, USA

Biliana Alexandrova-Kabadjova Banco de México, Mexico City, Mexico

Periklis Andritsos Faculty of Information, University of Toronto, Toronto, ON, Canada

Peter Auer Department of Information Technology, University of Leoben, Leoben, Austria

J. Andrew Bagnell Carnegie Mellon University, Pittsburgh, PA, USA

Michael Bain University of New South Wales, Sydney, NSW, Australia

Arindam Banerjee University of Minnesota, Minneapolis, MN, USA

Andrew G. Barto University of Massachusetts, Amherst, MA, USA

Rohan A. Baxter Australian Taxation Office, Sydney, NSW, Australia

Bettina Berendt KU Leuven, Leuven, Belgium

Indrajit Bhattacharya IBM India Research Laboratory, New Delhi, India

Mustafa Bilgic University of Maryland, College Park, MD, USA

Mauro Birattari Université Libre de Bruxelles, Brussels, Belgium

Hendrik Blockeel Katholieke Universiteit Leuven, Heverlee, Leuven, Belgium

Leiden Institute of Advanced Computer Science, Heverlee, Belgium

Shawn Bohn Pacific Northwest National Laboratory, Richland, WA, USA

Antal van den Bosch Centre for Language Studies, Radboud University, Nijmegen, The Netherlands

Luka Bradesko Jožef Stefan Institute, Ljubljana, Slovenia

Janez Brank Jožef Stefan Insitute, Ljubljana, Slovenia

Jürgen Branke University of Warwick, Coventry, UK

Pavel Brazdil LIAAD-INESC Tec/Faculdade de Economia, University of Porto, Porto, Portugal

Gavin Brown The University of Manchester, Manchester, UK

Ivan Bruha McMaster University, Hamilton, ON, Canada

Dariusz Brzezinski Institute of Computing Sciences, Poznan University of Technology, Poznan, Poland

Martin D. Buhmann Justus-Liebig University, Gießen, Germany

Wray L. Buntine Statistical Machine Learning Program, NICTA, Canberra, ACT, Australia

Faculty of Information Technology, Monash University, Clayton, VIC, Australia

Tibério Caetano Statistical Machine Learning Program, NICTA, Canberra, ACT, Australia

Nicola Cancedda Xerox Research Centre Europe, Meylan, France

Gail A. Carpenter Department of Mathematics & Center for Adaptive Systems, Boston University, Boston, MA, USA

John Case University of Delaware, Newark, DE, USA

Tonatiuh Peña Centeno German Center for Neurodegenerative Diseases, Banco de México, Mexico City, Mexico

Deepayan Chakrabarti Yahoo! Research, Sunnyvale, CA, USA

Philip K. Chan Florida Institute of Technology, Melbourne, FL, USA

Varun Chandola State University of New York at Buffalo, Buffalo, NY, USA

Zhiyuan Chen University of Illinois at Chicago, Chicago, IL, USA

Peter Christen Research School of Computer Science, The Australian National University, Canberra, ACT, Australia

Massimiliano Ciaramita Yahoo! Research Barcelona, Barcelona, Spain

Adam Coates Stanford University, Stanford, CA, USA

David Cohn Mountain View, CA, USA

Edinburgh, UK

David Corne Herriot-Watt University, Edinburgh, UK

Susan Craw Robert Gordon University, Aberdeen, UK

James Cussens University of York, Heslington, UK

Artur Czumaj University of Warwick, Coventry, UK

Walter Daelemans CLIPS University of Antwerp, Antwerpen, Belgium

Sanjoy Dasgupta University of California, San Diego, La Jolla, CA, USA

Gerald DeJong University of Illinois at Urbana, Urbana, IL, USA

Marco Dorigo Université Libre de Bruxelles, Brussels, Belgium

Kurt Driessens Maastricht University, Maastricht, The Netherlands

Chris Drummond National Research Council of Canada, Ottawa, ON, Canada

Lan Du Faculty of Information Technology, Monash University, Clayton, VIC, Australia

Yaakov Engel University of Alberta, Edmonton, AB, Canada

Scott E. Fahlman Carnegie Mellon University, Pittsburgh, PA, USA

Alan Fern Science, Oregon State University, Corvallis, OR, USA

Peter A. Flach Department of Computer Science, University of Bristol, Bristol, UK

Pierre Flener Department of Information Technology, Uppsala University, Uppsala, Sweden

Blaž Fortuna Jozef Stefan Institute, Ljubljana, Slovenia

Johannes Fürnkranz Knowledge Engineering Group, TU Darmstadt, Darmstadt, Deutschland

Department of Information Technology, University of Leoben, Leoben, Austria

Thomas Gärtner Fraunhofer IAIS, Schloss Birlinghoven, University of Bonn, Sankt Augustin, Germany

João Gama University of Porto, Porto, Portugal

Alma Lilia García-Almanza Directorate of Regulation and Supervision, Banco de México, Mexico City, Mexico

Gemma C. Garriga Universite Pierre et Marie Curie, Paris, France

Wulfram Gerstner Brain Mind Institute, Lausanne EPFL, Lausanne, Switzerland

Lise Getoor University of Maryland, College Park, MD, USA

Christophe Giraud-Carrier Department of Computer Science, Brigham Young University, Provo, UT, USA

Marko Grobelnik Artificial Intelligence Laboratory, Jožef Stefan Insitute, Ljubljana, Slovenia

Stephen Grossberg Center for Adaptive Systems, Graduate Program in Cognitive and Neural Systems, Department of Mathematics, Boston University, Boston, MA, USA

Eyke Hüllermeier Department of Computer Science, Paderborn University, Paderborn, Germany

Jiawei Han University of Illinois at Urbana-Champaign, Urbana, IL, USA

Julia Handl University of Manchester, Manchester, UK

Michael Harries Citrix Labs, Advanced Products Group, North Ryde, NSW, Australia

Jun He Aberystwyth University, Aberystwyth, UK

Bernhard Hengst University of New South Wales, Sydney, NSW, Australia

Tom Heskes Institute for Computing and Information Sciences, Radboud University Nijmegen, Nijmegen, The Netherlands

Geoffrey Hinton University of Toronto, Toronto, ON, Canada

James Hodson AI for Good Foundation, New York, NY, USA

Lawrence Holder Washington State University, Pullman, WA, USA

Tamás Horváth Fraunhofer IAIS, Schloss Birlinghoven, University of Bonn, Sankt Augustin, Germany

Phil Husbands Department of Informatics, Centre for Computational Neuroscience and Robotics, University of Sussex, Brighton, UK

Marcus Hutter Research School of Computer Science, Australian National University, Canberra, ACT, Australia

Christian Igel Department of Computer Science, University of Copenhagen, Copenhagen, Denmark

Sanjay Jain School of Computing, National University of Singapore, Singapore, Singapore

Szymon Jaroszewicz Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

Tommy R. Jensen Alpen-Adria-Universität Klagenfurt, Klagenfurt, Austria

Xin Jin PayPal Inc., San Jose, CA, USA

Antonis C. Kakas University of Cyprus, Nicosia, Cyprus

Subbarao Kambhampati Arizona State University, Tempe, AZ, USA

Anne Kao Boeing Phantom Works, Seattle, WA, USA

Samuel Kaski Helsinki University of Technology, Helsinki, Finland

Carlos Kavka University of Trieste, Trieste, Italy

James Kennedy U.S. Bureau of Labor Statistics, Washington, DC, USA

Eamonn Keogh University of California-Riverside, Riverside, CA, USA

Kristian Kersting Technische Universität Dortmund, Dortmund, Germany

Knowledge Discovery, Fraunhofer IAIS, Sankt Augustin, Germany

Joshua Knowles University of Manchester, Manchester, UK

Aleksander KoŁcz Microsoft One Microsoft Way, Redmond, WA, USA

Ron Kohavi Application Services Group, Microsoft, Bellevue, WA, USA

Kevin B. Korb Clayton School of Information Technology, Monash University, Clayton, VIC, Australia

Petra Kralj Novak Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia

Stefan Kramer Technische Universität München, Garching b. München, Germany

Krzysztof Krawiec Poznan University of Technology, Poznan, Poland

Vipin Kumar University of Minnesota, Minneapolis, MN, USA

Nicolas Lachiche University of Strasbourg, Strasbourg, France

Michail G. Lagoudakis Technical University of Crete, Chania, Greece

John Langford Microsoft Research, New York, NY, USA

Pier Luca Lanzi Politecnico di Milano, Milano, Italy

Nada Lavrač Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia

University of Nova Gorica, Nova Gorica, Slovenia

Gregor Leban Jozef Stefan Institute, Ljubljana, Slovenia

Christina Leslie Memorial Sloan Kettering Cancer Research Center, New York, NY, USA

Hang Li Huawei Technologies, Hong Kong, China

Shiau Hong Lim University of Illinois, Champaign, IL, USA

Charles X. Ling The University of Western Ontario, London, ON, Canada

Bin Liu Monash University, Clayton, VIC, Australia

Bing Liu University of Illinois at Chicago, Chicago, IL, USA

Huan Liu Arizona State University, Tempe, AZ, USA

John Lloyd The Australian National University, Canberra, ACT, Australia

Roger Longbotham Data and Decision Sciences Group, Microsoft, Redmond, WA, USA

Shie Mannor Israel Institute of Technology, Haifa, Israel

Serafín Martínez-Jaramillo Directorate of Financial System Risk Analysis, Banco de México, Mexico City, Mexico

Eric Martin University of New South Wales, Sydney, NSW, Australia

Stan Matwin University of Ottawa, Ottawa, ON, Canada

Polish Academy of Sciences, Warsaw, Poland

Julian McAuley Computer Science Department, University of California, San Diego, CA, USA

Statistical Machine Learning Program, NICTA, Canberra, ACT, Australia

Franziska Meier University of Southern California, Los Angeles, CA, USA

Prem Melville IBM T. J. Watson Research Center, Yorktown Heights, NY, USA

Pietro Michelucci Strategic Analysis, Inc., Arlington, VA, USA

Rada Mihalcea University of North Texas, Denton, TX, USA

Risto Miikkulainen Department of Computer Science, The University of Texas at Austin, Austin, TX, USA

Dunja Mladenić Artificial Intelligence Laboratory, Jožef Stefan Insitute, Ljubljana, Slovenia

Katharina Morik Technische Universität Dortmund, Dortmund, Germany

Jun Morimoto Advanced Telecommunication Research Institute International (ATR), Kyoto, Japan

Abdullah Mueen University California-Riverside, Riverside, CA, USA

Paul Munro University of Pittsburgh, Pittsburgh, PA, USA

Ion Muslea Language Weaver, Inc., Marina del Rey, CA, USA

Galileo Namata University of Maryland, College Park, MD, USA

Sriraam Natarajan Department of Computer Science, University of Wisconsin Medical School, Madison, WI, USA

School of Informatics and Computing, Indiana University, Bloomington, IN, USA

Andrew Y. Ng Computer Science Department, Stanford University, Stanford, CA, USA

Stanford University, Stanford, CA, USA

Siegfried Nijssen Katholieke Universiteit Leuven, Leuven, Belgium

William Stafford Noble Department of Genome Science/Department of Computer Science and Engineering, University of Washington, Seattle, WA, USA

Eirini Ntoutsi Leibniz Universität Hannover, Hannover, Germany

Ludwig Maximilians Universität München, Munich, Germany

Daniel Oblinger DARPA/IPTO, Arlington, VA, USA

Peter Orbanz Cambridge University, Cambridge, UK

Miles Osborne University of Edinburgh, Edinburgh, UK

Stefano Pacifico Jožef Stefan Institute, Ljubljana, Slovenia

C. David Page Department of Biostatistics and Medical Informatics, University of Wisconsin Medical School, Madison, WI, USA

Jonathan Patrick University of Ottawa, Ottawa, ON, Canada

Claudia Perlich IBM T.J. Watson Research Center, Yorktown Heights, NY, USA

Jan Peters Department of Empirical Inference, Max-Planck Institute for Intelligent Systems, Tübingen, Germany

Intelligent Autonomous Systems, Computer Science Department, Technische Universität Darmstadt, Darmstadt, Hessen, Germany

Max Planck Institute for Biological Cybernetics, Tübingen, Germany

Francesco Petruccione National Institute of Theoretical Physics (NITheP), KwaZulu-Natal, South Africa

Bernhard Pfahringer University of Waikato, Hamilton, New Zealand

Steve Poteet Boeing Phantom Works, Seattle, WA, USA

Pascal Poupart University of Waterloo, Waterloo, ON, Canada

Rob Powers Stanford University, Stanford, CA, USA

Cecilia M. Procopiuc AT&T Labs, NJ, USA

Martin L. Puterman University of British Columbia, Vancouver, BC, Canada

Lesley Quach Boeing Phantom Works, Seattle, WA, USA

Novi Quadrianto Department of Informatics, SMiLe CLiNiC, University of Sussex, Brighton, UK

Luc De Raedt Department of Computer Science, Katholieke Universiteit Leuven, Heverlee, Leuven, Belgium

Dev Rajnarayan NASA Ames Research Center, Moffett Field, CA, USA

Adwait Ratnaparkhi Yahoo!, Sunnyvale, CA, USA

Soumya Ray Case Western Reserve University, Cleveland, OH, USA

Mark Reid The Australian National University, Canberra, ACT, Australia

Jean-Michel Renders Xerox Research Centre Europe, Meylan, France

John Risch Pacific Northwest National Laboratory, Richland, WA, USA

Teemu Roos Department of Computer Science, Helsinki Institute for Information Technology, University of Helsinki, Helsinki, Finland

Nick Roy Massachusetts Institute of Technology, Cambridge, MA, USA

Lorenza Saitta Università del Piemonte Orientale, Alessandria, Italy

Yasubumi Sakakibara Keio University, Hiyoshi, Kohoku-ku, Japan

Claude Sammut The University of New South Wales, Sydney, NSW, Australia

Joerg Sander University of Alberta, Edmonton, AB, Canada

Statistical Machine Learning Group, NICTA, Canberra, ACT, Australia

Scott Sanner Statistical Machine Learning Group, NICTA, Canberra, ACT, Australia

Stefan Schaal Max Planck Institute for Intelligent Systems, Stuttgart, Germany

Computer Science, University of Southern California, Los Angeles, CA, USA

Ute Schmid Faculty of Information Systems and Applied Computer Science, University of Bamberg, Bamberg, Germany

Jürgen Schmidhuber The Swiss AI Lab, IDSIA, USI & SUPSI, Manno & Lugano, Switzerland

Maria Schuld Quantum Research Group, School of Chemistry & Physics, University of KwaZulu-Natal, Durban, South Africa

Stephen Scott University of Nebraska, Lincoln, NE, USA

Michele Sebag CNRS – INRIA – Université Paris-Sud, Orsay, France

Prithviraj Sen University of Maryland, College Park, MD, USA

Hanhuai Shan University of Minnesota, Minneapolis, MN, USA

Hossam Sharara University of Maryland, College Park, MD, USA

Viktoriia Sharmanska Department of Informatics, University of Sussex, SMiLe CLiNiC, Falmer, UK

Victor S. Sheng The University of Western Ontario, London, ON, Canada

Jelber Sayyad Shirabad University of Ottawa, Ottawa, ON, Canada

Yoav Shoham Stanford University, Stanford, CA, USA

Thomas R. Shultz McGill University, Montréal, QC, Canada

Ricardo Silva Centre for Computational Statistics and Machine Learning, University College London, London, UK

Vikas Sindhwani IBM T. J. Watson Research Center, Yorktown Heights, NY, USA

Moshe Sipper Ben-Gurion University, Beer-Sheva, Israel

William D. Smart Washington University in St. Louis, St. Louis, MO, USA

Carlos Soares LIAAD-INESC Porto L.A./Faculdade de Economia, University of Porto, Porto, Portugal

LIAAD-INESC Tec/Faculdade de Economia, University of Porto, Porto, Portugal

Christian Sohler University of Paderborn, Paderborn, Germany

Myra Spiliopoulou Otto-von-Guericke University-Magdeburg, Magdeburg, Germany

Thomas Stützle Université libre de Bruxelles (ULB), Brussels, Belgium

Janez Starc Jožef Stefan Institute, Ljubljana, Slovenia

Jerzy Stefanowski Institute of Computing Sciences, Poznan University of Technology, Poznan, Poland

Frank Stephan Department of Mathematics, National University of Singapore, Singapore, Singapore

Peter Stone Department of Computer Science, The University of Texas at Austin, Austin, TX, USA

Alexander L. Strehl Rütgers University, New Brunswick, NJ, USA

Jan Struyf Katholieke Universiteit Leuven, Heverlee, Leuven, Belgium

Prasad Tadepalli School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA

Jiliang Tang Michigan State University, East Lansing, MI, USA

Russ Tedrake Massachusetts Institute of Technology, Cambridge, MA, USA

Yee Whye Teh University College London, London, UK

Jon Timmis University of York, Heslington, North Yorkshire, UK

Jo-Anne Ting University of Edinburgh, Edinburgh, UK

Kai Ming Ting Federation University, Mount Helen, VIC, Australia

Ljupčo Todorovski University of Ljubljana, Ljubljana, Slovenia

Hannu Toivonen University of Helsinki, Helsinki, Finland

Luís Torgo University of Porto, Porto, Portugal

Panayiotis Tsaparas Department of Computer Science & Engineering, University of Ioannina, Ioannina, Greece

Paul E. Utgoff University of Massachusetts, Amherst, MA, USA

William Uther NICTA and The University of New South Wales, Sydney, NSW, Australia

Sethu Vijayakumar University of Edinburgh, Edinburgh, UK

University of Southern California, Los Angeles, CA, USA

Ricardo Vilalta Department of Computer Science, University of Houston, Houston, TX, USA

Michail Vlachos IBM Research, Zurich, Switzerland

Kiri L. Wagstaff Pasadena, CA, USA

Suhang Wang Arizona State University, Tempe, AZ, USA

Geoffrey I. Webb Faculty of Information Technology, Monash University, Victoria, Australia

R. Paul Wiegand University of Central Florida, Orlando, FL, USA

Eric Wiewiora University of California, Sydney, NSW, Australia

William E. Winkler US Census Bureau, Suitland, MD, USA

Anthony Wirth The University of Melbourne, Melbourne, VLC, Australia

Michael Witbrock Cycorp Inc, Austin, TX, USA

David Wolpert NASA Ames Research Center, Moffett Field, CA, USA

Santa Fe Institute, Santa Fe, NM, USA

Stefan Wrobel Fraunhofer IAIS, Schloss Birlinghoven, University of Bonn, Sankt Augustin, Germany

Jason Wu Boeing Phantom Works, Seattle, WA, USA

Zhao Xu Fraunhofer IAIS, Sankt Augustin, Germany

Ying Yang Australian Taxation Office, Box Hill, VIC, Australia

Sungwook Yoon MapR, San Jose, CA, USA

Thomas Zeugmann Hokkaido University, Sapporo, Japan

Xinhua Zhang NICTA, Australian National University, Canberra, ACT, Australia

School of Computer Science, Australian National University, Canberra, ACT, Australia

NICTA London Circuit, Canberra, ACT, Australia

Lei Zhang LinkedIn, San Francisco, CA, USA

Min-Ling Zhang School of Computer Science and Engineering, Southeast University, Nanjing, China

Fei Zheng Monash University, Sydney, NSW, Australia

Monash University, Clayton, Melbourne, VIC, Australia

Zhi-Hua Zhou National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

Xiaojin Zhu University of Wisconsin-Madison, Madison, WI, USA

Max Zimmermann Swedish Institute of Computer Science (SICS Swedish ICT), Kista, Sweden

A

A/B Testing

▶ Online Controlled Experiments and A/B Testing

Abduction

Antonis C. Kakas
University of Cyprus, Nicosia, Cyprus

Definition

Abduction is a form of reasoning, sometimes described as "deduction in reverse," whereby given a rule that "$A\ follows\ from\ B$" and the observed result of "A" we infer the condition "B" of the rule. More generally, given a theory, T, modeling a domain of interest and an observation, "A," we infer a hypothesis "B" such that the observation follows deductively from T augmented with "B." We think of "B" as a possible explanation for the observation according to the given theory that contains our rule. This new information and its consequences (or ramifications) according to the given theory can be considered as the result of a (or part of a) learning process based on the given theory and driven by the observations that are explained by abduction. Abduction can be combined with ▶ induction in different ways to enhance this learning process.

Motivation and Background

Abduction is, along with induction, a *synthetic* form of reasoning whereby it generates, in its explanations, new information not hitherto contained in the current theory with which the reasoning is performed. As such, it has a natural relation to learning, and in particular to *knowledge intensive learning*, where the new information generated aims to complete, at least partially, the current knowledge (or model) of the problem domain as described in the given theory.

Early uses of abduction in the context of machine learning concentrated on how abduction can be used as a theory revision operator for identifying where the current theory could be revised in order to accommodate the new learning data. This includes the work of Michalski (1993), Ourston and Mooney (1994), and Ade et al. (1994). Another early link of abduction to learning was given by the ▶ explanation based learning method (DeJong and Mooney 1986), where the abductive explanations of the learning data (training examples) are generalized to all cases. An extensive survey of the role of abduction in Machine Learning during this early period can be found in Bergadano et al. (2000).

Following this, it was realized (Flach and Kakas 2000) that the role of abduction in learning could be strengthened by linking it to induction, culminating in a hybrid integrated approach to learning where abduction and induction are tightly integrated to provide powerful learning frameworks such as the ones of Progol 5.0

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

(Muggleton and Bryant 2000) and HAIL (Ray et al. 2003). On the other hand, from the point of view of abduction as "inference to the best explanation" (Josephson and Josephson 1994) the link with induction provides a way to distinguish between different explanations and to select those explanations that give a better inductive generalization result.

A recent application of abduction, on its own or in combination with induction, is in Systems Biology where we try to model biological processes and pathways at different levels. This challenging domain provides an important development test-bed for these methods of knowledge intensive learning (see e.g., King et al. 2004; Papatheodorou et al. 2005; Ray et al. 2006; Tamaddoni-Nezhad et al. 2004; Zupan et al. 2003).

Structure of the Learning Task

Abduction contributes to the learning task by first explaining, and thus rationalizing, the training data according to a given and current model of the domain to be learned. These abductive explanations either form on their own the result of learning or they feed into a subsequent phase to generate the final result of learning.

Abduction in Artificial Intelligence

Abduction as studied in the area of Artificial Intelligence and the perspective of learning is mainly defined in a logic-based approach. Other approaches to abduction include set covering (See, e.g., Reggia 1983) or case-based explanation, (e.g., Leake 1995). The following explanation uses a logic-based approach.

Given a set of sentences T (a theory or model), and a sentence O (observation), the abductive task is the problem of finding a set of sentences H (abductive explanation for O) such that:

1. $T \cup H \models O$,
2. $T \cup H$ is consistent,

where $\models$ denotes the deductive entailment relation of the formal logic used in the representation of our theory and consistency refers also to the corresponding notion in this logic. The particular choice of this underlying formal framework of logic is in general a matter that depends on the problem or phenomena that we are trying to model. In many cases, this is based on ▸ first order predicate calculus, as, for example, in the approach of theory completion in Muggleton and Bryant (2000). But other logics can be used, e.g., the nonmonotonic logics of default logic or logic programming with negation as failure when the modeling of our problem requires this level of expressivity.

This basic formalization as it stands, does not fully capture the explanatory nature of the abductive explanation H in the sense that it necessarily conveys some reason why the observations hold. It would, for example, allow an observation O to be explained by itself or in terms of some other observations rather than in terms of some "deeper" reason for which the observation must hold according to the theory T. Also, as the above specification stands, the observation can be abductively explained by generating in H some new (general) theory completely unrelated to the given theory T. In this case, H does not account for the observations O according to the given theory T and in this sense it may not be considered as an explanation for O relative to T. For these reasons, in order to specify a "level" at which the explanations are required and to understand these relative to the given general theory about the domain of interest, the members of an explanation are normally restricted to belong to a special preassigned, domain-specific class of sentences called *abducible*.

Hence abduction, is typically applied on a model, T, in which we can separate two disjoint sets of predicates: the *observable* predicates and the *abducible* (*or open*) predicates. The basic assumption then is that our model T has reached a sufficient level of comprehension of the domain such that all the incompleteness of the model can be isolated (under some working hypotheses) in its abducible predicates. The observable predicates are assumed to be completely defined (in T) in terms of the abducible predicates and other background auxiliary predicates; any incompleteness in their representation comes from the incompleteness in the abducible predicates. In practice, the empirical observations that drive the learning task are described using the observable predicates. Observations are represented by formulae that refer only to the observable predicates (and possibly some background auxiliary predicates) typically by ground atomic facts on these observable predicates. The abducible predicates describe underlying (theoretical) relations in our model that are not observable directly but can, through the model T, bring about observable information.

The assumptions on the abducible predicates used for building up the explanations may be subject to restrictions that are expressed through *integrity constraints*. These represent additional knowledge that we have on our domain expressing general properties of the domain that remain valid no matter how the theory is to be extended in the process of abduction and associated learning. Therefore, in general, an *abductive theory* is a triple, denoted by $\langle T, A, \mathrm{IC}\rangle$, where T is the background theory, A is a set of abducible predicates, and IC is a set of integrity constraints. Then, in the definition of an abductive explanation given above, one more requirement is added:

3. $T \cup H$ satisfies IC.

The satisfaction of integrity constraints can be formally understood in several ways (see Kakas et al. 1992 and references therein). Note that the integrity constraints reduce the number of explanations for a set of observations filtering out those explanations that do not satisfy them. Based on this notion of abductive explanation a *credulous* form of abductive entailment is defined. Given an abductive theory, $T = \langle T, A, \mathrm{IC}\rangle$, and an observation O then, O is *abductively entailed* by T, denoted by $T \models_A O$, if there exists an abductive explanation of O in T.

This notion of abductive entailment can then form the basis of a coverage relation for learning in the face of incomplete information.

Abductive Concept Learning

Abduction allows us to reason in the face of incomplete information. As such when we have learning problems where the background data on the training examples is incomplete the use of abduction can enhance the learning capabilities.

Abductive concept learning (ACL) (Kakas and Riguzzi 2000) is a learning framework that allows us to learn from incomplete information and to later be able to classify new cases that again could be incompletely specified. Under ACL, we learn abductive theories, $\langle T, A, \mathrm{IC}\rangle$ with abduction playing a central role in the covering relation of the learning problem. The abductive theories learned in ACL contain both rules, in T, for the concept(s) to be learned as well as general clauses acting as integrity constraints in IC.

Practical problems that can be addressed with ACL: (1) concept learning from incomplete background data where some of the background predicates are incompletely specified and (2) concept learning from incomplete background data together with given integrity constraints that provide some information on the incompleteness of the data. The treatment of incompleteness through abduction is integrated within the learning process. This allows the possibility of learning more compact theories that can alleviate the problem of over fitting due to the incompleteness in the data. A specific subcase of these two problems and important third application problem of ACL is that of (3) multiple predicate learning, where each predicate is required to be learned from the incomplete data for the other predicates. Here the abductive reasoning can be used to suitably connect and integrate the learning of the different predicates. This can help to overcome some of the nonlocality difficulties of multiple predicate learning, such as order-dependence and global consistency of the learned theory.

ACL is defined as an extension of ▸ Inductive Logic Programming (ILP) where both the background knowledge and the learned theory are abductive theories. The central formal definition of ACL is given as follows where examples are atomic ground facts on the target predicate(s) to be learned.

Definition 1 (Abductive Concept Learning) Given

- A set of positive examples E^+
- A set of negative examples E^-
- An abductive theory $T = \langle P, A, I \rangle$ as background theory
- An hypothesis space $\mathcal{T} = \langle \mathcal{P}, \mathcal{I} \rangle$ consisting of a space of possible programs $\mathcal{P}$ and a space of possible constraints $\mathcal{I}$

Find

A set of rules $P' \in \mathcal{P}$ and a set of constraints $I' \in \mathcal{I}$ such that the new abductive theory $T' = \langle P \cup P', A, I \cup I' \rangle$ satisfies the following conditions

- $T' \models_A E^+$
- $\forall e^- \in E^-, T' \not\models_A e^-$

where E^+ stands for the conjunction of all positive examples.

An individual example e is said to be *covered* by a theory T' if $T' \models_A e$. In effect, this definition replaces the deductive entailment as the example coverage relation in the ILP problem with abductive entailment to define the ACL learning problem.

The fact that the conjunction of positive examples must be covered means that, for every positive example, there must exist an abductive explanation and the explanations for all the positive examples must be consistent with each other. For negative examples, it is required that no abductive explanation exists for any of them. ACL can be illustrated as follows.

Example 1 Suppose we want to learn the concept $father$. Let the background theory be $T = \langle P, A, \emptyset \rangle$ where:

$$P = \{parent(john, mary), male(john), parent(david, steve), parent(kathy, ellen), female(kathy)\},$$
$$A = \{male, female\}.$$

Let the training examples be:

$$E^+ = \{father(john, mary), father(david, steve)\},$$
$$E^- = \{father(kathy, ellen), father(john, steve)\}.$$

In this case, a possible hypotheses $T' = \langle P \cup P', A, I' \rangle$ learned by ACL would consist of

$$P' = \{father(X,Y) \leftarrow parent(X,Y), male(X)\},$$
$$I' = \{\leftarrow male(X), female(X)\}.$$

This hypothesis satisfies the definition of ACL because:

1. $T' \models_A father(john, mary), father(david, steve)$ with $\Delta = \{male(david)\}$.
2. $T' \not\models_A father(kathy, ellen)$, as the only possible explanation for this goal, namely $\{male(kathy)\}$ is made inconsistent by the learned integrity constraint in I'.
3. $T' \not\models_A father(john, steve)$, as this has no possible abductive explanations.

Hence, despite the fact that the background theory is incomplete (in its abducible predicates), ACL can find an appropriate solution to the learning problem by suitably extending the background theory with abducible assumptions. Note that the learned theory without the integrity constraint would not satisfy the definition of ACL, because there would exist an abductive explanation for the negative example $father(kathy, ellen)$, namely $\Delta^- = \{male(kathy)\}$. This explanation is prohibited in the complete theory by the learned constraint together with the fact $female(kathy)$.

The algorithm and learning system for ACL is based on a decomposition of this problem into two subproblems: (1) learning the rules in P' together with appropriate explanations for the training examples and (2) learning integrity constraints driven by the explanations generated in the first part. This decomposition allows ACL to be developed by combining the two IPL settings of explanatory (predictive) learning and confirmatory (descriptive) learning. In fact, the first subproblem can be seen as a problem of learning from entailment, while the second subproblem as a problem of learning from interpretations.

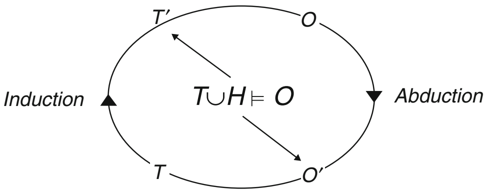
Abduction and Induction

The utility of abduction in learning can be enhanced significantly when this is integrated with *induction*. Several approaches for synthesizing abduction and induction in learning have been developed, e.g., Ade and Denecker (1995), Muggleton and Bryant (2000), Yamamoto (1997), and Flach and Kakas (2000). These approaches aim to develop techniques for knowledge intensive learning with complex background theories. One problem to be faced by purely inductive techniques, is that the training data on which the inductive process operates, often contain gaps and inconsistencies. The general idea is that abductive reasoning can feed information into the inductive process by using the background theory for inserting new hypotheses and removing inconsistent data. Stated differently, abductive inference is used to complete the training data with hypotheses about missing or inconsistent data that explain the example or training data, using the background theory. This process gives alternative possibilities for assimilating and generalizing this data.

Induction is a form of synthetic reasoning that typically generates knowledge in the form of new general rules that can provide, either directly, or indirectly through the current theory T that they extend, new interrelationships between the predicates of our theory that can include, unlike abduction, the observable predicates and even in some cases new predicates. The inductive hypothesis thus introduces new, hitherto unknown, links between the relations that we are studying thus allowing new predictions on the observable predicates that would not have been possible before from the original theory under any abductive explanation.

An inductive hypothesis, H, extends, like in abduction, the existing theory T to a new theory $T'=T \cup \ H$, but now H provides new links between observables and nonobservables that was missing or incomplete in the original theory T. This is particularly evident from the fact that induction can be performed even with an empty given theory T, using just the set of observations. The observations specify incomplete (usually extensional) knowledge about the observable predicates, which we try to *generalize* into new knowledge. In contrast, the generalizing effect of abduction, if at all present, is much more limited. With the given current theory T, that abduction always needs to refer to, we implicitly restrict the generalizing power of abduction as we require that the basic model of our domain remains that of T. Induction has a stronger and genuinely new generalizing effect on the observable predicates than abduction. While the purpose of abduction is to extend the theory with an explanation and then reason with it, thus enabling the generalizing potential of the given theory T, in induction the purpose is to extend the given theory to a new theory, which can provide new possible observable consequences.

This complementarity of abduction and induction – abduction providing explanations from the theory while induction generalizes to form new parts of the theory – suggests a basis for their integration within the context of theory formation and theory development. A *cycle of integration* of abduction and induction (Flach and Kakas 2000) emerges that is suitable for the task of incremental modeling (Fig. 1). Abduction is used to transform (and in some sense normalize) the observations to information on the abducible predicates. Then, induction takes this as input and tries to generalize this information to general



Abduction, Fig. 1 The cycle of abductive and inductive knowledge development. The cycle is governed by the "equation" $T \cup H \models O$, where T is the current theory, O the observations triggering theory development, and H the new knowledge generated. On the left-hand side we have induction, its output feeding into the theory T for later use by abduction on the right; the abductive output in turn feeds into the observational data O' for later use by induction, and so on rules for the abducible predicates now treating these as observable predicates for its own purposes. The cycle can then be repeated by adding the learned information on the abducibles back in the model as new partial information on the incomplete abducible predicates. This will affect the abductive explanations of new observations to be used again in a subsequent phase of induction. Hence, through this cycle of integration the abductive explanations of the observations are added to the theory, not in the (simple) form in which they have been generated, but in a generalized form given by a process of induction on these.

A simple example, adapted from Ray et al. (2003), that illustrates this cycle of integration of abduction and induction is as follows. Suppose that our current model, T, contains the following rule and background facts:

$$sad(X) \leftarrow tired(X), poor(X),$$

$$tired(oli), tired(ale), tired(kr),$$

$$academic(oli), academic(ale), academic(kr),$$

$$student(oli), lecturer(ale), lecturer(kr),$$

where the only observable predicate is $sad/1$.

Given the observations $O = \{sad(ale), sad(kr), not\ \ sad(oli)\}$ can we improve our model? The incompleteness of our model resides in the predicate $poor$. This is the only abducible predicate in our model. Using abduction we can explain the observations O via the explanation:

$$E = \{poor(ale), poor(kr), not\ poor(oli)\}.$$

Subsequently, treating this explanation as training data for inductive generalization we can generalize this to get the rule:

$$poor(X) \leftarrow lecturer(X)$$

thus (partially) defining the abducible predicate $poor$ when we extend our theory with this rule.

This combination of abduction and induction has recently been studied and deployed in several ways within the context of ILP. In particular, *inverse entailment* (Muggleton and Bryant 2000) can be seen as a particular case of integration of abductive inference for constructing a “bottom” clause and inductive inference to generalize it. This is realized in Progol 5.0 and applied to several problems including the discovery of the function of genes in a network of metabolic pathways (King et al. 2004), and more recently to the study of inhibition in metabolic networks (Tamaddoni-Nezhad et al. 2006, 2004). In Moyle (2000), an ILP system called ALECTO, integrates a phase of *extraction-case abduction* to transform each case of a training example to an abductive hypothesis with a phase of induction that generalizes these abductive hypotheses. It has been used to learn robot navigation control programs by completing the specific domain knowledge required, within a general theory of planning that the robot uses for its navigation (Moyle 2002).

The development of these initial frameworks that realize the cycle of integration of abduction and induction prompted the study of the problem of *completeness* for finding any hypotheses H that satisfies the basic task of finding a consistent hypothesis H such that $T \cup H \models O$ for a given theory T, and observations O. Progol was found to be incomplete (Yamamoto 1997) and several new frameworks of integration of abduction and induction have been proposed such as SOLDR (Ito and Yamamoto 1998), CF-induction (Inoue 2001), and HAIL (Ray et al. 2003). In particular, HAIL has demonstrated that one of the main reasons for the incompleteness of Progol is that in its cycle of integration of abduction and induction, it uses a very restricted form of abduction. Lifting some of these restrictions, through the employment of methods from abductive logic programming (Kakas et al. 1992), has allowed HAIL to solve a wider class of problems. HAIL has been extended to a framework, called XHAIL (Ray 2009), for learning nonmonotonic ILP, allowing it to be applied to learn Event Calculus theories for action description (Alrajeh et al. 2009) and complex scientific theories for systems biology (Ray and Bryant 2008).

Applications of this integration of abduction and induction and the cycle of knowledge development can be found in the recent proceedings of the Abduction and Induction in Artificial Intelligence workshops in 2007 (Flach and Kakas 2009) and 2009 (Ray et al. 2009).

Abduction in Systems Biology

Abduction has found a rich field of application in the domain of systems biology and the declarative modeling of computational biology. In a project called, Robot scientist (King et al. 2004), Progol 5.0 was used to generate abductive hypotheses about the function of genes. Similarly, learning the function of genes using abduction has been studied in GenePath (Zupan et al. 2003) where experimental genetic data is explained in order to facilitate the analysis of genetic networks. Also in Papatheodorou et al. (2005) abduction is used to learn gene interactions and genetic pathways from microarray experimental data. Abduction and its integration with induction has been used in the study of inhibitory effect of toxins in metabolic networks (Tamaddoni-Nezhad et al. 2004, 2006) taking into account also the temporal variation that the inhibitory effect can have. Another bioinformatics application of abduction (Ray et al. 2006) concerns the modeling of human immunodeficiency virus (HIV) drug resistance and using this in order to assist medical practitioners in the selection of antiretroviral drugs for patients infected with HIV. Also, the recently developed frameworks of XHAIL and CF-induction have been applied to several problems in systems biology, see e.g., Ray (2009), Ray and Bryant (2008), and Doncescu et al. (2007), respectively. Finally, the recent book edited by Cerro and Inoue (2014) on the logical modeling of biological systems contains several articles on the application of abduction in systems biology.

Cross-References

- ▶ Explanation-Based Learning
- ▶ Inductive Logic Programming

Recommended Reading

Ade H, Denecker M (1995) AILP: abductive inductive logic programming. In: Mellish CS (ed) IJCAI. Morgan Kaufmann, San Francisco, pp 1201–1209

Ade H, Malfait B, Raedt LD (1994) Ruth: an ILP theory revision system. In: ISMIS94. Springer, Berlin

Alrajeh D, Ray O, Russo A, Uchitel S (2009) Using abduction and induction for operational requirements elaboration. J Appl Logic 7(3):275–288

Bergadano F, Cutello V, Gunetti D (2000) Abduction in machine learning. In: Gabbay D, Kruse R (eds) Handbook of defeasible reasoning and uncertainty management systems, vol 4. Kluver Academic Press, Dordrecht, pp 197–229

del Cerro LF, Inoue K (eds) (2014) Logical modeling of biological systems. Wiley/ISTE, Hoboken/London

DeJong G, Mooney R (1986) Explanation-based learning: an alternate view. Mach Learn 1:145–176

Doncescu A, Inoue K, Yamamoto Y (2007) Knowledge based discovery in systems biology using cf-induction. In: Okuno HG, Ali M (eds) IEA/AIE. Springer, Heidelberg, pp 395–404

Flach P, Kakas A (2000) Abductive and inductive reasoning: background and issues. In: Flach PA, Kakas AC (eds) Abductive and inductive reasoning. Pure and applied logic. Kluwer, Dordrecht

Flach PA, Kakas AC (eds) (2009) Abduction and induction in artificial intelligence [special issue]. J Appl Logic 7(3):251

Inoue K (2001) Inverse entailment for full clausal theories. In: LICS-2001 workshop on logic and learning

Ito K, Yamamoto A (1998) Finding hypotheses from examples by computing the least generlisation of bottom clauses. In: Proceedings of discovery science'98. Springer, Berlin, pp 303–314

Josephson J, Josephson S (eds) (1994) Abductive inference: computation, philosophy, technology. Cambridge University Press, New York

Kakas A, Kowalski R, Toni F (1992) Abductive logic programming. J Logic Comput 2(6):719–770

Kakas A, Riguzzi F (2000) Abductive concept learning. New Gener Comput 18:243–294

King R, Whelan K, Jones F, Reiser P, Bryant C, Muggleton S et al (2004) Functional genomic hypothesis generation and experimentation by a robot scientist. Nature 427:247–252

Leake D (1995) Abduction, experience and goals: a model for everyday abductive explanation. J Exp Theor Artif Intell 7:407–428

Michalski RS (1993) Inferential theory of learning as a conceptual basis for multistrategy learning. Mach Learn 11:111–151

Moyle S (2002) Using theory completion to learn a robot navigation control program. In: Proceedings of the 12th international conference on inductive logic programming. Springer, Berlin, pp 182–197

Moyle SA (2000) An investigation into theory completion techniques in inductive logic programming. PhD thesis, Oxford University Computing Laboratory, University of Oxford

Muggleton S (1995) Inverse entailment and Progol. New Gener Comput 13:245–286

Muggleton S, Bryant C (2000) Theory completion using inverse entailment. In: Proceedings of the tenth international workshop on inductive logic programming (ILP-00). Springer, Berlin, pp 130–146

Ourston D, Mooney RJ (1994) Theory refinement combining analytical and empirical methods. Artif Intell 66:311–344

Papatheodorou I, Kakas A, Sergot M (2005) Inference of gene relations from microarray data by abduction. In: Proceedings of the eighth international conference on logic programming and non-monotonic reasoning (LPNMR'05), vol 3662. Springer, Berlin, pp389–393

Ray O (2009) Nonmonotonic abductive inductive learning. J Appl Logic 7(3):329–340

Ray O, Antoniades A, Kakas A, Demetriades I (2006) Abductive logic programming in the clinical management of HIV/AIDS. In: Brewka G, Coradeschi S, Perini A, Traverso P (eds) Proceedings of the 17th European conference on artificial intelligence. Frontiers in artificial intelligence and applications, vol 141. IOS Press, Amsterdam, pp 437–441

Ray O, Broda K, Russo A (2003) Hybrid abductive inductive learning: a generalisation of Progol. In: Proceedings of the 13th international conference on inductive logic programming. Lecture notes in artificial intelligence, vol 2835. Springer, Berlin, pp 311–328

Ray O, Bryant C (2008) Inferring the function of genes from synthetic lethal mutations. In: Proceedings of the second international conference on complex, intelligent and software intensive systems. IEEE Computer Society, Washington, DC, pp 667–671

Ray O, Flach PA, Kakas AC (eds) (2009) Abduction and induction in artificial intelligence. In: Proceedings of IJCAI 2009 workshop

Reggia J (1983) Diagnostic experts systems based on a set-covering model. Int J Man-Mach Stud 19(5):437–460

Tamaddoni-Nezhad A, Chaleil R, Kakas A, Muggleton S (2006) Application of abductive ILP to learning metabolic network inhibition from temporal data. Mach Learn 64(1–3):209–230

Tamaddoni-Nezhad A, Kakas A, Muggleton S, Pazos F (2004) Modelling inhibition in metabolic pathways through abduction and induction. In: Proceedings of the 14th international conference on inductive logic programming. Springer, Berlin, pp 305–322

Yamamoto A (1997) Which hypotheses can be found with inverse entailment? In: Proceedings of the seventh international workshop on inductive logic programming. Lecture notes in artificial intelligence, vol 1297. Springer, Berlin, pp 296–308

Zupan B, Bratko I, Demsar J, Juvan P, Halter J, Kuspa A et al (2003) Genepath: a system for automated construction of genetic networks from mutant data. Bioinformatics 19(3):383–389

Absolute Error Loss

▶ Mean Absolute Error

Accuracy

Definition

Accuracy refers to a measure of the degree to which the predictions of a model matches the reality being modeled. The term *accuracy* is often applied in the context of ▶ classification models. In this context, *accuracy* = P($\lambda(X) = Y$), where XY is a joint distribution and the classification model λ is a function $X \rightarrow Y$. Sometimes, this quantity is expressed as a percentage rather than a value between 0.0 and 1.0.

The accuracy of a model is often assessed or estimated by applying it to test data for which the ▶ labels (Y values) are known. The accuracy of a classifier on test data may be calculated as *number of correctly classified objects/total number of objects*. Alternatively, a smoothing function may be applied, such as a ▶ Laplace estimate or an *m*-estimate.

Accuracy is directly related to ▶ error rate, such that *accuracy* = 1.0 – *error rate* (or when expressed as a percentage, *accuracy* = 100 – *error rate*).

Cross-References

▶ Confusion Matrix
▶ Mean Absolute Error
▶ Model Evaluation
▶ Resubstitution Estimate

ACO

▶ Ant Colony Optimization

Actions

In a ▶ Markov decision process, *actions* are the available choices for the decision-maker at any given *decision epoch*, in any given *state*.

Active Learning

David Cohn
Mountain View, CA, USA
Edinburgh, UK

Definition

The term *Active Learning* is generally used to refer to a learning problem or system where the learner has some role in determining on what data it will be trained. This is in contrast to *Passive Learning*, where the learner is simply presented with a ▶ training set over which it has no control. Active learning is often used in settings where obtaining ▶ labeled data is expensive or time-consuming; by sequentially identifying which examples are most likely to be useful, an active learner can sometimes achieve good performance, using far less ▶ training data than would otherwise be required.

Structure of Learning System

In many machine learning problems, the training data are treated as a fixed and given part of the problem definition. In practice, however, the training data are often not fixed beforehand. Rather, the learner has an opportunity to play a role in deciding what data will be acquired for training. This process is usually referred to as "active learning," recognizing that the learner is an active participant in the training process.

The typical goal in active learning is to select training examples that best enable the learner to minimize its loss on future test cases. There are many theoretical and practical results demonstrating that, when applied properly, active learning can greatly reduce the number of training examples, and even the computational effort required for a learner to achieve good generalization.

A toy example that is often used to illustrate the utility of active learning is that of learning a threshold function over a one-dimensional interval. Given $+/-$ labels for N points drawn uniformly over the interval, the expected error between the true value of the threshold and any learner's best guess is bounded by $O(1/N)$. Given the opportunity to sequentially select the position of points to be labeled, however, a learner can pursue a binary search strategy, obtaining a best guess that is within $O(1/2^N)$ of the true threshold value.

This toy example illustrates the sequential nature of example selection that is a component of most (but not all) active learning strategies: the learner makes use of initial information to discard parts of the solution space, and to focus future data acquisition on distinguishing parts that are still viable.

Related Problems

The term "active learning" is usually applied in supervised learning settings, though there are many related problems in other branches of machine learning and beyond. The "exploration" component of the "exploration/exploitation" strategy in reinforcement learning is one such example. The learner *must* take actions to gain information, and must decide what actions will give him/her the information that will best minimize future loss. A number of fields of Operations Research predate and parallel machine learning work on active learning, including Decision Theory (North 1968), Value of Information Computation, Bandit problems (Robbins 1952), and Optimal Experiment Design (Fedorov 1972; Box and Draper 1987).

Active Learning Scenarios

When active learning is used for classification or regression, there are three common settings: *constructive* active learning, *pool-based* active learning, and *stream-based* active learning (also called *selective sampling*).

Constructive Active Learning

In constructive active learning, the learner is allowed to propose arbitrary points in the input space as examples to be labeled. While this in theory gives the learner the most power to explore, it is often not practical. One obstacle is the observation that most learning systems train on only a reduced representation of the instances they are presented with: text classifiers on bags of words (rather than fully-structured text) and speech recognizers on formants (rather than raw audio). While a learning system may be able to identify what pattern of formants would be most informative to label, there is no reliable way to generate audio that a human could recognize (and label) from the desired formants alone.

Pool-Based Active Learning

Pool-based active learning (McCallum and Nigam 1998) is popular in domains such as text classification and speech recognition where unlabeled data are plentiful and cheap, but labels are expensive and slow to acquire. In pool-based active learning, the learner may not propose arbitrary points to label, but instead has access to a set of unlabeled examples, and is allowed to select which of them to request labels for.

A special case of pool-based learning is transductive active learning, where the test distribution is exactly the set of unlabeled examples. The goal then is to sequentially select and label a small number of examples that will best allow predicting the labels of those points that remain unlabeled.

A theme that is common to both constructive and pool-based active learning is the principle of sequential experimentation. Information gained from early queries allows the learner to focus its search on portions of the domain that are most likely to give it additional information on subsequent queries.

Stream-Based Active Learning

Stream-based active learning resembles pool-based learning in many ways, except that the learner only has access to the unlabeled instances as a stream; when an instance arrives, the learner must decide whether to ask for its label or let it go.

Other Forms of Active Learning

By virtue of the broad definition of active learning, there is no real limit on the possible settings for framing it. Angluin's early work on learning regular sets (Angluin 1987) employed a "counterexample" oracle: the learner would propose a solution, and the oracle would either declare it correct, or divulge a counterexample – an instance on which the proposed and true solutions disagreed. Jin and Si (2003) describe a Bayesian method for selecting informative items to recommend when learning a collaborative filtering model, and Steck and Jaakkola (2002) describe a method best described as *unsupervised* active learning to build Bayesian networks in large domains.

While most active learning work assumes that the cost of obtaining a label is independent of the instance to be labeled, there are many scenarios where this is not the case. A mobile robot taking surface measurements must first travel to the point it wishes to sample, making distant points more expensive than nearby ones. In some cases, the cost of a query (e.g., the difficulty of traveling to a remote point to sample it) may not even be known until it is made, requiring the learner to learn a model of that as well. In these situations, the sequential nature of active learning is greatly accentuated, and a learner faces the additional challenges of planning under uncertainty (see "Greedy vs. Batch Active Learning," below).

Common Active Learning Strategies

1. *Version space partitioning*. The earliest practical active learning work (Ruff and Dietterich

1989; Mitchell 1982) explicitly relied on ▶ version space partitioning. These approaches tried to select examples on which there was maximal disagreement between hypotheses in the current version space. When such examples were labeled, they would invalidate as large a portion of the version space as possible. A limitation of explicit version space approaches is that, in underconstrained domains, a learner may waste its effort differentiating portions of the version space that have little effect on the classifier's predictions, and thus on its error.

2. *Query by Committee* (Seung et al. 1992). In query by committee, the experimenter trains an ensemble of models, either by selecting randomized starting points (e.g., in the case of a neural network) or by bootstrapping the training set. Candidate examples are scored based on disagreement among the ensemble models – examples with high disagreement indicate areas in the sample space that are underdetermined by the training data, and therefore potentially valuable to label. Models in the ensemble may be looked at as samples from the version space; picking examples where these models disagree is a way of splitting the version space.
3. *Uncertainty sampling* (Lewis and Gail 1994). Uncertainty sampling is a heuristic form of statistical active learning. Rather than sampling different points in the version space by training multiple learners, the learner itself maintains an explicit model of uncertainty over its input space. It then selects for labeling those examples on which it is least confident. In classification and regression problems, uncertainty contributes directly to expected loss (as the variance component of the "error = bias + variance" decomposition), so that gathering examples where the learner has greatest uncertainty is often an effective loss-minimization heuristic. This approach has also been found effective for non-probabilistic models, by simply selecting examples that lie near the current decision boundary. For some learners, such as support vector machines, this heuristic can be shown to be an approximate partitioning of the learner's version space (Tong and Koller 2001).
4. *Loss minimization* (Cohn 1996). Uncertainty sampling can stumble when parts of the learner's domain are inherently noisy. It may be that, regardless of the number of samples labeled in some neighborhood, it will remain impossible to accurately predict these. In these cases, it would be desirable to not only model the learner's uncertainty over arbitrary parts of its domain, but also to model what effect labeling any future example is expected to have on that uncertainty. For some learning algorithms it is feasible to explicitly compute such estimates (e.g., for locally-weighted regression and mixture models, these estimates may be computed in closed form). It is, therefore, practical to select examples that directly minimize the expected loss to the learner, as discussed below under "Statistical Active Learning."

Statistical Active Learning

Uncertainty sampling and direct loss minimization are two examples of *statistical* active learning. Both rely on the learner's ability to statistically model its own uncertainty. When learning with a statistical model, such as a linear regressor or a mixture of Gaussians (Dasgupta 1999), the objective is usually to find model parameters that minimize some form of expected loss. When active learning is applied to such models, it is natural to also select training data so as to minimize that same objective. As statistical models usually give us estimates on the probability of (as yet) unknown values, it is often straightforward to turn this machinery upon itself to assist in the active learning process (Cohn 1996). The process is usually as follows:

1. Begin by requesting labels for a small random subsample of the examples x_1, x_2, K, $x_n x$ and fit our model to the labeled data.
2. For any x in our domain, a statistical model lets us estimate both the conditional expec-

tation $\hat{y}(x)$ and $\sigma^2_{\hat{y}(x)}$, the variance of that expectation. We estimate our current loss by drawing a new random sample of unlabeled data, and computing the averaged $\sigma^2_{\hat{y}(x)}$.

3. We now consider a candidate point $\tilde{x}$, and ask what reduction in loss we would obtain if we had labeled it $\tilde{y}$. If we knew its label with certainty, we could simply add the point to the training set, retrain, and compute the new expected loss. While we do not know the true $\tilde{y}$, we could, in theory, compute the new expected loss for every possible $\tilde{y}$ and average those losses, weighting them by our model's estimate of $p(\tilde{y}|\tilde{y})$. In practice, this is normally unfeasible; however, for some statistical models, such as locally-weighted regression and mixtures of Gaussians, we can compute the distribution of $p(\tilde{y}|\tilde{y})$ and its effect on $\sigma^2_{\hat{y}(x)}$ in closed form (Cohn 1996).
4. Given the ability to estimate the expected effect of obtaining label $\tilde{y}$ for candidate $\tilde{x}$, we repeat this computation for a sample of M candidates, and then request a label for the candidate with the largest expected decrease in loss. We add the newly-labeled example to our training set, retrain, and begin looking at candidate points to add on the next iteration.

The Need for Reference Distributions

Step (2) above illustrates a complication that is unique to active learning approaches. Traditional "passive" learning usually relies on the assumption that the distribution over which the learner will be tested is the same as the one from which the training data were drawn. When the learner is allowed to select its own training data, it still needs some form of access to the distribution of data on which it will be tested. A pool-based or stream-based learner can use the pool or stream as a proxy for that distribution, but if the learner is allowed (or required) to construct its own examples, it risks wasting all its effort on resolving portions of the solution space that are of no interest to the problem at hand.

A Detailed Example: Statistical Active Learning with LOESS

LOESS (Cleveland et al. 1988) is a simple form of locally-weighted regression using a kernel function. When asked to predict the unknown output y corresponding to a given input x, LOESS computes a ▶ linear regression over known (x, y) pairs, in which it gives pair (x_i, y_i) weight according to the proximity of x_i to x. We will write this weighting as a kernel function, $K(x_i, x)$, or simplify it to k_i when there is no chance of confusion.

In the active learning setting, we will assume that we have a large supply of unlabeled examples drawn from the test distribution, along with labels for a small number of them. We wish to label a small number more so as to minimize the mean squared error (MSE) of our model. MSE can be decomposed into two terms: squared ▶ bias and variance. If we make the (inaccurate but simplifying) assumption that LOESS is approximately unbiased for the problem at hand, minimizing MSE reduces to minimizing the variance of our estimates.

Given n labeled pairs, and a prediction to make for input x, LOESS computes the following covariance statistics around x:

$$\mu_x = \frac{\Sigma_i k_i x_i}{n}, \quad \sigma^2_x = \frac{\Sigma_i k_i (x_i - \mu_x)^2}{n},$$

$$\sigma_{xy} = \frac{\Sigma_i k_i (x_i - \mu_x)(y_i - \mu_y)}{n}$$

$$\mu_y = \frac{\Sigma_i k_i y_i}{n}, \quad \sigma^2_y = \frac{\Sigma_i k_i (y_i - \mu_y)^2}{n},$$

$$\sigma^2_{y|x} = \sigma^2_y - \frac{\sigma_{xy}}{\sigma^2_x}$$

We can combine these to express the conditional expectation of y (our estimate) and its variance as:

$$\hat{y} = \mu_y + \frac{\sigma_{xy}}{\sigma^2_x}(x - \mu_x), \sigma^2_{\hat{y}} = \frac{\sigma^2_{y|x}}{n^2}$$

$$\times \left(\sum_i k_i^2 + \frac{(x - \mu_x)^2}{\sigma^2_x} \sum_i k_i^2 \frac{(x_i - \mu_x)^2}{\sigma^2_x} \right).$$

Our proxy for model error is the variance of our prediction, integrated over the test distribution $\left\langle \sigma_{\hat{y}}^2 \right\rangle$. As we have assumed a pool-based setting in which we have a large number of unlabeled examples from that distribution, we can simply compute the above variance over a sample from the pool, and use the resulting average as our estimate.

To perform statistical active learning, we want to compute how our estimated variance will change if we add an (as yet unknown) label $\tilde{y}$ for an arbitrary $\tilde{x}$. We will write this new expected variance as $\left\langle \tilde{\sigma}_{\hat{y}}^2 \right\rangle$. While we do not *know* what value $\tilde{y}$ will take, our model gives us an estimated mean $\hat{y}(\tilde{x})$ and variance $\sigma_{\hat{y}(\tilde{x})}^2$ for the value, as above. We can add this "distributed" y value to LOESS just as though it were a discrete one, and compute the resulting expectation $\left\langle \tilde{\sigma}_{\hat{y}}^2 \right\rangle$ in closed form. Defining $\tilde{k}$ as $K(\tilde{x}, x)$, we write:

$$\left\langle \tilde{\sigma}_{\hat{y}}^2 \right\rangle = \frac{\left\langle \tilde{\sigma}_{y|x}^2 \right\rangle}{(n+\tilde{k})^2} \left(\sum_i k_i^2 + \tilde{k}^2 + \frac{(x-\tilde{\mu}_x)^2}{\tilde{\sigma}_x^2} \right. \\ \left. \times \left(\sum_i k_i^2 \frac{(x_i-\tilde{\mu}_x)^2}{\tilde{\sigma}_x^2} + \tilde{k}^2 \frac{(\tilde{x}-\tilde{\mu}_x)^2}{\tilde{\sigma}_x^2} \right) \right),$$

where the component expectations are computed as follows:

$$\left\langle \tilde{\sigma}_{y|x}^2 \right\rangle = \left\langle \tilde{\sigma}_y^2 \right\rangle - \frac{\left\langle \tilde{\sigma}_{xy}^2 \right\rangle}{\tilde{\sigma}_x^2},$$

$$\left\langle \tilde{\sigma}_y^2 \right\rangle = \frac{n\sigma_y^2}{n+\tilde{k}} + \frac{n\tilde{k}(\sigma_{y1\overline{x}}^2 + (\hat{y}(\tilde{x})-\mu_y)^2)}{(n+\tilde{k})^2},$$

$$\tilde{\mu}_x = \frac{n\mu_x + \tilde{k}\tilde{x}}{n+\tilde{k}},$$

$$\left\langle \tilde{\sigma}_{xy} \right\rangle = \frac{n\sigma_{xy}}{n+\tilde{k}} + \frac{n\tilde{k}(\tilde{x}-\mu_x)(\hat{y}(\tilde{x})-\mu_y)}{(n+\tilde{k})^2},$$

$$\tilde{\sigma}_x^2 = \frac{n\sigma_x^2}{n+\tilde{k}} + \frac{n\tilde{k}(\tilde{x}-\mu_x)^2}{(n+\tilde{k})^2},$$

$$\left\langle \tilde{\sigma}_{xy}^2 \right\rangle = \left\langle \tilde{\sigma}_{xy} \right\rangle^2 + \frac{n^2\tilde{k}^2\sigma_{y|\overline{x}}^2(\tilde{x}-\mu_x)^2}{(n+\tilde{k})^4}.$$

Greedy Versus Batch Active Learning

It is also worth pointing out that virtually all active learning work relies on greedy strategies – the learner estimates what single example best achieves its objective, requests that one, retrains, and repeats. In theory, it is possible to plan some number of queries ahead, asking what point is best to label now, given that N-1 more labeling opportunities remain. While such strategies have been explored in Operations Research for very small problem domains, their computational requirements make this approach unfeasible for problems of the size typically encountered in machine learning.

There are cases where retraining the learner after every new label would be prohibitively expensive, or where access to labels is limited by the number of iterations as well as by the total number of labels (e.g., for a finite number of clinical trials). In this case, the learner may select a set of examples to be labeled on each iteration. This batch approach, however, is only useful if the learner is able to identify a set of examples whose expected contributions are non-redundant, which substantially complicates the process.

Cross-References

▸ Active Learning Theory

Recommended Reading

Angluin D (1987) Learning regular sets from queries and counterexamples. Inf Comput 75(2):87–106

Angluin D (1988) Queries and concept learning. Mach Learn 2:319–342

Box GEP, Draper N (1987) Empirical model-building and response surfaces. Wiley, New York

Cleveland W, Devlin S, Gross E (1988) Regression by local fitting. J Econom 37:87–114

Cohn D, Atlas L, Ladner R (1990) Training connectionist networks with queries and selective sampling. In: Touretzky D (ed) Advances in neural information processing systems. Morgan Kaufmann, San Mateo

Cohn D, Ghahramani Z, Jordan MI (1996) Active learning with statistical models. J Artif Intell Res 4:129–145. http://citeseer.ist.psu.edu/321503.html

Dasgupta S (1999) Learning mixtures of Gaussians. Found Comput Sci 634–644

Fedorov V (1972) Theory of optimal experiments. Academic Press, New York

Kearns M, Li M, Pitt L, Valiant L (1987) On the learnability of Boolean formulae. In: Proceedings of the 19th annual ACM conference on theory of computing. ACM Press, New York, pp 285–295

Lewis DD, Gail WA (1994) A sequential algorithm for training text classifiers. In: Proceedings of the 17th annual international ACM SIGIR conference, Dublin, pp 3–12

McCallum A, Nigam K (1998) Employing EM and pool-based active learning for text classification. In: Machine learning: proceedings of the fifteenth international conference (ICML'98), Madison, pp 359–367

North DW (1968) A tutorial introduction to decision theory. IEEE Trans Syst Sci Cybern 4(3)

Pitt L, Valiant LG (1988) Computational limitations on learning from examples. J ACM (JACM) 35(4):965–984

Robbins H (1952) Some aspects of the sequential design of experiments. Bull Am Math Soc 55:527–535

Ruff R, Dietterich T (1989) What good are experiments? In: Proceedings of the sixth international workshop on machine learning, Ithaca

Seung HS, Opper M, Sompolinsky H (1992) Query by committee. In: Proceedings of the fifth workshop on computational learning theory. Morgan Kaufmann, San Mateo, pp 287–294

Steck H, Jaakkola T (2002) Unsupervised active learning in large domains. In: Proceeding of the conference on uncertainty in AI. http://citeseer.ist.psu.edu/steck02unsupervised.html

Active Learning Theory

Sanjoy Dasgupta
University of California, San Diego, La Jolla, CA, USA

Definition

The term *active learning* applies to a wide range of situations in which a learner is able to exert some control over its source of data. For instance, when fitting a regression function, the learner may itself supply a set of data points at which to measure response values, in the hope of reducing the variance of its estimate. Such problems have been studied for many decades under the rubric of *experimental design* (Chernoff 1972; Fedorov 1972). More recently, there has been substantial interest within the machine learning community in the specific task of actively learning binary classifiers. This task presents several fundamental statistical and algorithmic challenges, and an understanding of its mathematical underpinnings is only gradually emerging. This brief survey will describe some of the progress that has been made so far.

Learning from Labeled and Unlabeled Data

In the machine learning literature, the task of learning a classifier has traditionally been studied in the framework of *supervised learning*. This paradigm assumes that there is a training set consisting of data points x (from some set $\mathcal{X}$) and their labels y (from some set $\mathcal{Y}$), and the goal is to learn a function $f : \mathcal{X} \to \mathcal{Y}$, that will accurately predict the labels of data points arising in the future. Over the past 50 years, tremendous progress has been made in resolving many of the basic questions surrounding this model, such as "how many training points are needed to learn an accurate classifier?"

Although this framework is now fairly well understood, it is a poor fit for many modern learning tasks because of its assumption that all training points automatically come labeled. In practice, it is frequently the case that the raw, abundant, easily obtained form of data is *unlabeled*, whereas labels must be explicitly procured and are expensive. In such situations, the reality is that the learner starts with a large pool of unlabeled points and must then strategically decide which ones it wants labeled: how best to spend its limited budget.

Example: Speech recognition. When building a speech recognizer, the unlabeled training data consists of raw speech samples, which are very easy to collect: just walk around with a microphone. For all practical purposes, an unlimited quantity of such samples can be obtained. On the other hand, the “label” for each speech sample is a segmentation into its constituent phonemes, and producing even one such label requires substantial human time and attention. Over the past decades, research labs and the government have expended an enormous amount of money, time, and effort on creating labeled datasets of English speech. This investment has paid off, but our ambitions are inevitably moving past what these datasets can provide: we would now like, for instance, to create recognizers for other languages, or for English in specific contexts. Is there some way to avoid more painstaking years of data labeling, to somehow leverage the easy availability of raw speech so as to significantly reduce the number of labels needed? This is the hope of active learning.

Some early results on active learning were in the *membership query* model, where the data is assumed to be *separable* (that is, some hypothesis h perfectly classifies all points) and the learner is allowed to query the label of *any* point in the input space $\mathcal{X}$ (rather than being constrained to a prespecified unlabeled set), with the goal of eventually returning the perfect hypothesis h^*. There is a significant body of beautiful theoretical work in this model (Angluin 2001), but early experiments ran into some telling difficulties. One study (Baum and Lang 1992) found that when training a neural network for handwritten digit recognition, the queries synthesized by the learner were such bizarre and unnatural images that they were impossible for a human to classify. In such contexts, the membership query model is of limited practical value; nonetheless, many of the insights obtained from this model carry over to other settings (Hanneke 2007a).

We will fix as our standard model one in which the learner is *given* a source of unlabeled data, rather than being able to generate these points himself. Each point has an associated label, but the label is initially *hidden*, and there is a cost for revealing it. The hope is that an accurate classifier can be found by querying just a few labels, much fewer than would be required by regular supervised learning.

How can the learner decide which labels to probe? One option is to select the query points at random, but it is not hard to show that this yields the same label complexity as supervised learning. A better idea is to choose the query points *adaptively*: for instance, start by querying some random data points to get a rough sense of where the decision boundary lies, and then gradually refine the estimate of the boundary by specifically querying points in its immediate vicinity. In other words, ask for the labels of data points whose particular positioning makes them especially informative. Such strategies certainly sound good, but can they be fleshed out into practical algorithms? And if so, do these algorithms work well in the sense of producing good classifiers with fewer labels than would be required by supervised learning?

On account of the enormous practical importance of active learning, there are a wide range of algorithms and techniques already available, most of which resemble the aggressive, adaptive sampling strategy just outlined, and many of which show promise in experimental studies. However, a big problem with this kind of sampling is that very quickly the set of labeled points no longer reflects the underlying data distribution. This makes it hard to show that the classifiers learned have good statistical properties (for instance, that they converge to an optimal classifier in the limit of infinitely many labels). This survey will only discuss methods that have proofs of statistical well-foundedness, and whose label complexity can be explicitly analyzed.

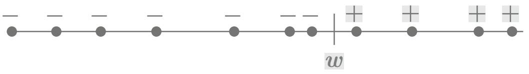
Motivating Examples

We will start by looking at a few examples that illustrate the enormous potential of active learning and that also make it clear why analyses of this new model require concepts and intuitions that are fundamentally different from those that have already been developed for supervised learning.

Example: Thresholds on the Line

Suppose the data lie on the real line, and the available classifiers are simple thresholding functions, $\mathcal{H} = \{h_w : w \in \mathbb{R}\}$:

$$h_w(x) = \begin{cases} +1 & \text{if } x \geq w \\ -1 & \text{if } x < w \end{cases}$$



To make things precise, let us denote the (unknown) underlying distribution on the data $(X, Y) \in \mathbb{R} \times \{+1, -1\}$ by $\mathbb{P}$, and let us suppose that we want a hypothesis $h \in \mathcal{H}$ whose error with respect to $\mathbb{P}$, namely $\text{err}_{\mathbb{P}} = \mathbb{P}(h(X) \neq Y)$, is at most some ϵ. How many labels do we need?

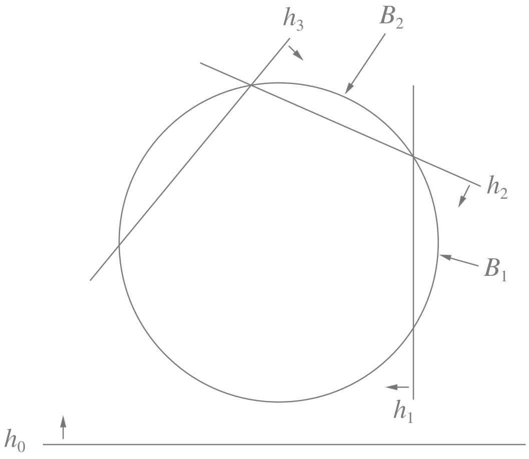
In supervised learning, such issues are well understood. The standard machinery of sample complexity (using VC theory) tells us that if the data are *separable* – that is, if they can be perfectly classified by some hypothesis in $\mathcal{H}$ – then we need approximately $1/\epsilon$ random labeled examples from $\mathbb{P}$, and it is enough to return any classifier consistent with them.

Now suppose we instead draw $1/\epsilon$ *unlabeled* samples from $\mathbb{P}$. If we lay these points down on the line, their hidden labels are a sequence of −s followed by a sequence of +s, and the goal is to discover the point w at which the transition occurs. This can be accomplished with a simple binary search which asks for just log $1/\epsilon$ labels: first ask for the label of the median point; if it is +, move to the 25th percentile point, otherwise move to the 75th percentile point; and so on. Thus, for this hypothesis class, active learning gives an *exponential* improvement in the number of labels needed, from $1/\epsilon$ to just $\log 1/\epsilon$. For instance, if supervised learning requires a million labels, active learning requires just $\log 1{,}000{,}000 \approx 20$, literally!

It is a tantalizing possibility that even for more complicated hypothesis classes $\mathcal{H}$, a sort of generalized binary search is possible. A natural next step is to consider linear separators in *two* dimensions.

Example: Linear Separators in $\mathbb{R}^2$

Let $\mathcal{H}$ be the hypothesis class of linear separators in $\mathbb{R}^2$, and suppose the data is distributed according to some density supported on the perimeter of the unit circle. It turns out that the positive results of the one-dimensional case do not generalize: there are some target hypotheses in $\mathcal{H}$ for which $\Omega(1/\epsilon)$ labels are needed to find a classifier with error rate less than ϵ, no matter what active learning scheme is used.



Active Learning Theory, Fig. 1 $\mathbb{P}$ is supported on the circumference of a circle. Each B_i is an arc of probability mass ϵ

To see this, consider the following possible target hypotheses (Fig. 1):

- h_0: all points are positive.
- $h_i (1 \leq i \leq 1/\epsilon)$: all points are positive except for a small slice B_i of probability mass ϵ.

The slices B_i are explicitly chosen to be disjoint, with the result that $\Omega(1/\epsilon)$ labels are needed to distinguish between these hypotheses. For instance, suppose nature chooses a target hypothesis at random from among the h_i, $1 \leq i \leq 1/\epsilon$. Then, to identify this target with probability at least 1/2, it is necessary to query points in at least (about) half the B_is.

Thus for these particular target hypotheses, active learning offers little improvement in sample complexity over regular supervised learning. What about other target hypotheses in $\mathcal{H}$, for instance those in which the positive and negative regions are more evenly balanced? It is quite easy (Dasgupta 2005) to devise an active learning scheme which asks for $O(\min\{1/i(h), 1/\epsilon\}) + O(\log 1/\epsilon)$ labels, where $i(h) = \min\{$positive

Pool-based active learning

```
Get a set of unlabeled points $U \subset \mathcal{X}$
Repeat until satisfied:
  Pick some $x \in U$ to label
Return a hypothesis $h \in \mathcal{H}$
```

Stream-based active learning

```
Repeat for $t = 0, 1, 2, \ldots$:
  Choose a hypothesis $h_t \in \mathcal{H}$
  Receive an unlabeled point $x \in \mathcal{X}$
  Decide whether to query its label
```

Active Learning Theory, Fig. 2 Models of pool-and stream-based active learning. The data are draws from an underlying distribution $\mathbb{P}_X$, and hypotheses h are evaluated by $\mathrm{err}_{\mathbb{P}}(h)$. If we want to get this error below ϵ, how many labels are needed, as a function of ϵ?

mass of h, negative mass of $h\}$. Thus even within this simple hypothesis class, the label complexity can run anywhere from $O(\log 1/\epsilon)$ to $\Omega(1/\epsilon)$, depending on the specific target hypothesis!

Example: An Overabundance of Unlabeled Data

In our two previous examples, the amount of unlabeled data needed was $O(\log 1/\epsilon)$, exactly the usual sample complexity of supervised learning. But it is sometimes helpful to have significantly more unlabeled data than this. In Dasgupta (2005), a distribution $\mathbb{P}$ is described for which if the amount of unlabeled data is small (below any prespecified threshold), then the number of labels needed to learn the target linear separator is $\Omega(1/\epsilon)$; whereas if the amount of unlabeled data is much larger, then only $O(\log 1/\epsilon)$ labels are needed. This is a situation where most of the data distribution is fairly uninformative while a miniscule fraction is highly informative. A lot of unlabeled data is needed in order to get even a few of the informative points.

The Sample Complexity of Active Learning

We will think of the unlabeled points $x_1, \ldots, x_n$ as being drawn i.i.d. from an underlying distribution $\mathbb{P}_X$ on $\mathcal{X}$ (namely, the marginal of the distribution $\mathbb{P}$ on $\mathcal{X} \times \mathcal{Y}$), either all at once (a *pool*) or one at a time (a *stream*). The learner is only allowed to query the labels of points in the pool/stream; that is, it is restricted to "naturally occurring" data points rather than synthetic ones (Fig. 2). It returns a hypothesis $h \in \mathcal{H}$ whose quality is measured by its error rate, $\mathrm{err}_{\mathbb{P}}(h)$

In regular supervised learning, it is well known that if the VC dimension of $\mathcal{H}$ is d, then the number of labels that will with high probability ensure $\mathrm{err}_{\mathbb{P}}(h) \leq \epsilon$ is roughly $O(d/\epsilon)$ if the data is separable and $O(d/\epsilon^2)$ otherwise (Haussler 1992); various logarithmic terms are omitted here. For active learning, it is clear from the examples above that the VC dimension alone does not adequately characterize label complexity. Is there a different combinatorial parameter that does?

Generic Results for Separable Data

For separable data, it is possible to give upper and lower bounds on label complexity in terms of a special parameter known as the *splitting index* (Dasgupta et al. 2005). This is merely an existence result: the algorithm needed to realize the upper bound is intractable because it involves explicitly maintaining an ϵ-cover (a coarse approximation) of the hypothesis class, and the size of this cover is in general exponential in the VC dimension. Nevertheless, it does give us an idea of the kinds of label complexity we can hope to achieve.

Example Suppose the hypothesis class consists of intervals on the real line: $\mathcal{X} = \mathbb{R}$ and $\mathcal{H} = \{h_{a,b} : a, b \in \mathbb{R}\}$, where $h_{a,b}(x) = \mathbf{1}(a \leq x \leq b)$. Using the splitting index, the label complexity of active learning is seen to be $\tilde{\Theta}(\min\{1/\mathbb{P}_X([a,b]), 1/\epsilon\} + \log 1/\epsilon)$ when the target hypothesis is $h_{a,b}$ (Dasgupta 2005). Here the $\tilde{\Theta}$ notation is used to suppress logarithmic terms.

Example Suppose $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{H}$ consists of linear separators through the origin. If $\mathbb{P}_X$ is the uniform distribution on the unit sphere, the number of labels needed to learn a hypothesis of error $\leq \epsilon$ is just $\tilde{\Theta}(d \log 1/\epsilon)$, exponentially smaller than the $\tilde{O}(d/\epsilon)$ label complexity of supervised learning. If $\mathbb{P}_X$ is not the uniform distribution but is everywhere within a multiplicative factor $\lambda > 1$ of it, then the label complexity becomes $\tilde{O}((d \log 1/\epsilon) \log^2 \lambda)$, provided the amount of unlabeled data is increased by a factor of λ^2 (Dasgupta 2005).

These results are very encouraging, but the question of an *efficient* active learning algorithm remains open. We now consider two approaches.

Mildly Selective Sampling

The label complexity results mentioned above are based on querying maximally informative points. A less aggressive strategy is to be *mildly selective*, to query all points except those that are quite clearly uninformative. This is the idea behind one of the earliest generic active learning schemes (Cohn et al. 1994). Data points $x_1, x_2, \ldots$ arrive in a stream, and for each one the learner makes a spot decision about whether or not to request a label. When x_t arrives, the learner behaves as follows.

- Determine whether both possible labelings, $(x_t, +)$ and $(x_t, -)$, are consistent with the labeled examples seen so far.
- If so, ask for the label y_t. Otherwise set y_t to be the unique consistent label.

Fortunately, the check required for the first step can be performed efficiently by making two calls to a supervised learner. Thus this is a very simple and elegant active learning scheme, although as one might expect, it is suboptimal in its label complexity (Balcan et al. 2007). Interestingly, there is a parameter called the *disagreement coefficient* that characterizes the label complexity of this scheme and also of some other mildly selective learners (Friedman 2009; Hanneke 2007b).

In practice, the biggest limitation of the algorithm above is that it assumes the data are separable. Recent results have shown how to remove this assumption (Balcan et al. 2006; Dasgupta et al. 2007) and to accommodate classification loss functions other than $0-1$ loss (Beygelzimer et al. 2009). Variants of the disagreement coefficient continue to characterize label complexity in the agnostic setting (Beygelzimer et al. 2009; Dasgupta et al. 2007).

A Bayesian Model

The *query by committee* algorithm (Seung et al. 1992) is based on a Bayesian view of active learning. The learner starts with a prior distribution on the hypothesis space, and is then exposed to a stream of unlabeled data. Upon receiving x_t, the learner performs the following steps.

- Draw two hypotheses h, h' at random from the posterior over $\mathcal{H}$.
- If $h(x_t) \neq h'(x_t)$ then ask for the label of x_t and update the posterior accordingly.

This algorithm queries points that substantially shrink the posterior, while at the same time taking account of the data distribution. Various theoretical guarantees have been shown for it (Freund et al. 1997); in particular, in the case of linear separators with a uniform data distribution, it achieves a label complexity of $O(d \log 1/\epsilon)$, the best possible.

Sampling from the posterior over the hypothesis class is, in general, computationally prohibitive. However, for linear separators with a uniform prior, it can be implemented efficiently using random walks on convex bodies (Gilad-Bachrach et al. 2005).

Other Work

In this survey, I have touched mostly on active learning results of the greatest generality, those that apply to arbitrary hypothesis classes. There is also a significant body of more specialized results.

- Efficient active learning algorithms for specific hypothesis classes.

This includes an online learning algorithm for linear separators that only queries some of the points and yet achieves similar regret bounds to algorithms that query all the points (Cesa-Bianchi et al. 2004). The label complexity of this method is yet to be characterized.

- Algorithms and label bounds for linear separators under the uniform data distribution.

  This particular setting has been amenable to mathematical analysis. For separable data,it turns out that a variant of the perceptron algorithm achieves the optimal $O(d \log 1/\epsilon)$ label complexity (Dasgupta 2005). A simple algorithm is also available for the agnostic setting (Balcan et al. 2007).

Conclusion

The theoretical frontier of active learning is mostly an unexplored wilderness. Except for a few specific cases, we do not have a clear sense of how much active learning can reduce label complexity: whether by just a constant factor, or polynomially, or exponentially. The fundamental statistical and algorithmic challenges involved, together with the huge practical importance of the field, make active learning a particularly rewarding terrain for investigation.

Cross-References

▶ Active Learning

Recommended Reading

Angluin D (2001) Queries revisited. In: Proceedings of the 12th international conference on algorithmic learning theory, Washington, DC, pp 12–31

Balcan M-F, Beygelzimer A, Langford J (2006) Agnostic active learning. In: International conference on machine learning. ACM Press, New York, pp 65–72

Balcan M-F, Broder A, Zhang T (2007) Margin based active learning. In: Conference on learning theory, San Diego, pp 35–50

Baum EB, Lang K (1992) Query learning can work poorly when a human oracle is used. In: International joint conference on neural networks, Baltimore

Beygelzimer A, Dasgupta S, Langford J (2009) Importance weighted active learning. In: International conference on machine learning. ACM Press, New York, pp 49–56

Cesa-Bianchi N, Gentile C, Zaniboni L (2004) Worst-case analysis of selective sampling for linear-threshold algorithms. In: Advances in neural information processing systems

Chernoff H (1972) Sequential analysis and optimal design. CBMS-NSF regional conference series in applied mathematics, vol 8. SIAM, Philadelphia

Cohn D, Atlas L, Ladner R (1994) Improving generalization with active learning. Mach Learn 15(2):201–221

Dasgupta S (2005) Coarse sample complexity bounds for active learning. Advances in neural information processing systems. Morgan Kaufmann, San Mateo

Dasgupta S, Kalai A, Monteleoni C (2005) Analysis of perceptron-based active learning. In: 18th annual conference on learning theory, Bertinoro, pp 249–263

Dasgupta S, Hsu DJ, Monteleoni C (2007) A general agnostic active learning algorithm. Advances in neural information processing systems

Fedorov VV (1972) Theory of optimal experiments (trans: Studden WJ, Klimko EM). Academic Press, New York

Freund Y, Seung S, Shamir E, Tishby N (1997) Selective sampling using the query by committee algorithm. Mach Learn J 28:133–168

Friedman E (2009) Active learning for smooth problems. In: Conference on learning theory, Montreal, pp 343–352

Gilad-Bachrach R, Navot A, Tishby N (2005) Query by committeee made real. Advances in neural information processing systems

Hanneke S (2007a) Teaching dimension and the complexity of active learning. In: Conference on learning theory, San Diego, pp 66–81

Hanneke S (2007b) A bound on the label complexity of agnostic active learning. In: International conference on machine learning, Corvallis, pp 353–360

Haussler D (1992) Decision-theoretic generalizations of the PAC model for neural net and other learning applications. Inf Comput 100(1):78–150

Seung HS, Opper M, Sompolinsky H (1992) Query by committee. In: Conference on computational learning theory, Victoria, pp 287–294

Adaboost

Adaboost is an ▶ ensemble learning technique, and the most well-known of the ▶ Boosting family of algorithms. The algorithm trains models sequentially, with a new model trained at each round. At the end of each round, mis-classified examples are identified and have their emphasis increased in a new training set which is then fed back into the start of the next round, and a new model is trained. The idea is that subsequent models should be able to compensate for errors made by earlier models. See ▶ ensemble learning for full details.

Adaptive Control Processes

▶ Bayesian Reinforcement Learning

Adaptive Learning

▶ Metalearning

Adaptive Real-Time Dynamic Programming

Andrew G. Barto
University of Massachusetts, Amherst, MA, USA

Synonyms

ARTDP

Definition

Adaptive Real-Time Dynamic Programming (ARTDP) is an algorithm that allows an agent to improve its behavior while interacting over time with an incompletely known dynamic environment. It can also be viewed as a heuristic search algorithm for finding shortest paths in incompletely known stochastic domains. ARTDP is based on ▶ Dynamic Programming (DP), but unlike conventional DP, which consists of off-line algorithms, ARTDP is an on-line algorithm because it uses agent behavior to guide its computation. ARTDP is adaptive because it does not need a complete and accurate model of the environment but learns a model from data collected during agent-environment interaction. When a good model is available, ▶ Real-Time Dynamic Programming (RTDP) is applicable, which is ARTDP without the model-learning component.

Motivation and Background

RTDP combines strengths of heuristic search and DP. Like heuristic search – and unlike conventional DP – it does not have to evaluate the entire state space in order to produce an optimal solution. Like DP – and unlike most heuristic search algorithms – it is applicable to nondeterministic problems. Additionally, RTDP's performance as an ▶ anytime algorithm is better than conventional DP and heuristic search algorithms. ARTDP extends these strengths to problems for which a good model is not initially available.

In artificial intelligence, control engineering, and operations research, many problems require finding a policy (or control rule) that determines how an agent (or controller) should generate actions in response to the states of its environment (the controlled system). When a "cost" or a "reward" is associated with each step of the agent's behavior, policies can be compared according to how much cost or reward they are expected to accumulate over time.

The usual formulation for problems like this in the discrete-time case is the ▶ Markov Decision Process (MDP). The objective is to find a policy that minimizes (maximizes) a measure of the total cost (reward) over time, assuming that the agent–environment interaction can begin in any of the possible states. In other cases, there is a designated set of "start states" that is much smaller than the entire state set (e.g., the initial board configuration in a board game). In these cases, any given policy only has to be defined for the set of states that can be reached from the starting states when the agent is using that policy.

The rest of the states will never arise when that policy is being followed, so the policy does not need to specify what the agent should do in those states.

ARTDP and RTDP exploit situations where the set of states reachable from the start states is a small subset of the entire state space. They can dramatically reduce the amount of computation needed to determine an optimal policy for the relevant states as compared with the amount of computation that a conventional DP algorithm would require to determine an optimal policy for all the states. These algorithms do this by focussing computation around simulated behavioral experiences (if there is a model available capable of simulating these experiences), or around real behavioral experiences (if no model is available).

RTDP and ARTDP were introduced by Barto et al. (1995). The starting point was the novel observation by Bradtke that Korf's Learning Real-Time A* heuristic search algorithm (Korf 1990) is closely related to DP. RTDP generalizes Learning Real-Time A* to stochastic problems. ARTDP is also closely related to Sutton's Dyna system (Sutton 1990) and Jalali and Ferguson's (1989) Transient DP. Theoretical analysis relies on the theory of Asynchronous DP as described by Bertsekas and Tsitsiklis (1989).

ARTDP and RTDP are ▸ model-based reinforcement learning algorithms, so called because they take advantage of an environment model, unlike ▸ model-free reinforcement learning algorithms such as ▸ Q-Learning and Sarsa.

Structure of Learning System

Backup Operations

A basic step of many DP and RL algorithms is a *backup operation*. This is an operation that updates a current estimate of the *cost* of an MDP's state. (We use the cost formulation instead of reward to be consistent with the original presentation of the algorithms. In the case of rewards, this would be called the *value* of a state and we would maximize instead of minimize.) Suppose X is the set of MDP states. For each state $x \in X$, $f(x)$, the cost of state x, gives a measure (which varies with different MDP formulations) of the total cost the agent is expected to incur over the future if it starts in x. If $f_k(x)$ and $f_{k+1}(x)$, respectively, denote the estimate of $f(x)$ before and after a backup, a typical backup operation applied to x looks like this:

$$f_{k+1}(x) = min_{a \in A}[c_x(a) + \sum_{y \in X} p_{xy}(a) f_k(fv)],$$

where A is the set of possible agent actions, $c_x(a)$ is the immediate cost the agent incurs for performing action a in state x, and $p_{xy}(a)$ probability that the environment makes a transition from state x to state y as a result of the agent's action a. This backup operation is associated with the DP algorithm known as ▸ value iteration. It is also the backup operation used by RTDP and ARTDP.

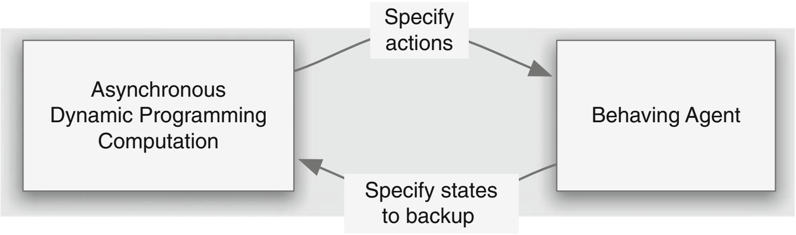
Conventional DP algorithms consist of successive "sweeps" of the state set. Each sweep consists of applying a backup operation to each state. Sweeps continue until the algorithm converges to a solution. Asynchronous DP, which underlies RTDP and ARTDP, does not use systematic sweeps. States can be chosen in any way whatsoever, and as long as backups continue to be applied to all states (and some other conditions are satisfied), the algorithm will converge. RTDP is an instance of asynchronous DP in which the states chosen for backups are determined by the agent's behavior.

The backup operation above is *model-based* because it uses known rewards and transition probabilities, and the values of all the states appear on the right-hand-side of the equation. In contrast, a *sample backup* uses the value of just one sample successor state. RTDP and ARTDP are like RL algorithms in that they rely on real or simulated behavioral experience, but unlike many (but not all) RL algorithms, they use full backups like DP.

Off-Line Versus On-Line

A conventional DP algorithm typically executes off-line. When applied to finding an optimal policy for an MDP, this means that the DP algorithm executes to completion before its result (an optimal policy) is used to control the agent's behavior. The sweeps of DP sequentially "visit" the states of the MDP, performing a backup operation on each state. But it is important not to confuse these visits with the behaving agent's visits to states: the agent is not yet behaving while the off-line DP computation is being done. Hence, the agent's behavior has no influence on the DP computation. The same is true for off-line asynchronous DP.

RTDP is an on-line, or "real-time," algorithm. It is an asynchronous DP computation that executes *concurrently* with the agent's behavior so that the agent's behavior can influence the DP computation. Further, the concurrently executing DP computation can influence the agent's behavior. The agent's visits to states directs the "visits" to states made by the concurrent asynchronous DP computation. At the same time, the action performed by the agent is the action specified by the policy corresponding to the latest results of the DP computation: it is the "greedy" action with respect to the current estimate of the cost function.



In the simplest version of RTDP, when a state is visited by the agent, the DP computation performs the model-based backup operation given above on that same state. In general, for each step of the agent's behavior, RTDP can apply the backup operation to each of an arbitrary set of states, provided that the agent's current state is included. For example, at each step of behavior, a limited-horizon look-ahead search can be conducted from the agent's current state, with the backup operation applied to each of the states generated in the search. Essentially, RTDP is an asynchronous DP computation with the computational effort focused along simulated or actual behavioral trajectories.

Learning A Model

ARTDP is the same as RTDP except that (1) an environment model is updated using any on-line model-learning, or system identification, method, (2) the current environment model is used in performing the RTDP backup operations, and (3) the agent has to perform exploratory actions occasionally instead of always greedy actions as in RTDP. This last step is essential to ensure that the environment model eventually converges to the correct model. If the state and action sets are finite, the simplest way to learn a model is to keep counts of the number of times each transition occurs for each action and convert these frequencies to probabilities, thus forming the maximum-likelihood model.

Summary of Theoretical Results

When RTDP and ARTDP are applied to *stochastic optimal path problems*, one can prove that under certain conditions they converge to optimal policies without the need to apply backup operations to all the states. Indeed, is some problems, only a small fraction of the states need to be visited. A stochastic optimal path problem is an MDP with a nonempty set of start states and a nonempty set of goal states. Each transition until a goal state is reached has a nonnegative immediate cost, and once the agent reaches a goal state, it stays there and thereafter incurs zero cost. Each episode of agent experience begins with a start state. An optimal policy is one that minimizes the cost of every state, i.e., minimizes $f(x)$ for all states x. Under some relatively mild conditions, every optimal policy is guaranteed to eventually reach a goal state.

A state x is *relevant* if a start state s and an optimal policy exist such that x can be reached from s when the agent uses that policy. If we could somehow know which states are relevant, we could restrict DP to just these states and obtain an optimal policy. But this is not possible because knowing which states are relevant requires knowledge of optimal policies, which is what one is seeking. However, under certain conditions, without requiring repeated visits to all the irrelevant states, RTDP produces a policy that is optimal for all the relevant states. The conditions are that (1) the initial cost of every goal state is zero, (2) there exists at least one policy that guarantees that a goal state will be reached with probability one from any start state, (3) all immediate costs for transitions from non-goal states are strictly positive, and (4) none of the initial costs are larger than the actual costs. This result is proved in Barto et al. (1995) by combining aspects of Korf's (1990) proof for LRTA* with results for asynchronous DP.

Special Cases and Extensions

A number of special cases and extensions of RTDP have been developed that improve performance over the basic version. Some examples are as follows. Bonet and Geffner's (2003a) Labeled RTDP labels states that have already been "solved," allowing faster convergence than RTDP. Feng et al. (2003) proposed Symbolic RTDP, which selects a set of states to update at each step using symbolic model-checking techniques. The RTDP convergence theorem still applies because this is a special case of RTDP. Smith and Simmons (2006) developed Focused RTDP that maintains a priority value for each state to better direct search and produce faster convergence. Hansen and Zilberstein's (2001) LAO* uses some of the same ideas as RTDP to produce a heuristic search algorithm that can find solutions with loops to non-deterministic heuristic search problems. Many other variants are possible. Extending ARTDP instead of RTDP in all of these ways would produce analogous algorithms that could be used when a good model is not available.

Cross-References

- ▶ Anytime Algorithm
- ▶ Approximate Dynamic Programming
- ▶ Reinforcement Learning

Recommended Reading

Barto A, Bradtke S, Singh S (1995) Learning to act using real-time dynamic programming. Artif Intell 72(1–2):81–138

Bertsekas D, Tsitsiklis J (1989) Parallel and distributed computation: numerical methods. Prentice-Hall, Englewood Cliffs

Bonet B, Geffner H (2003a) Labeled RTDP: improving the convergence of real-time dynamic programming. In: Proceedings of the 13th international conference on automated planning and scheduling (ICAPS-2003), Trento

Bonet B, Geffner H (2003b) Faster heuristic search algorithms for planning with uncertainty and full feedback. In: Proceedings of the international joint conference on artificial intelligence (IJCAI-2003), Acapulco

Feng Z, Hansen E, Zilberstein S (2003) Symbolic generalization for on-line planning. In: Proceedings of the 19th conference on uncertainty in artificial intelligence, Acapulco

Hansen E, Zilberstein S (2001) LAO*: a heuristic search algorithm that finds solutions with loops. Artif Intell 129:35–62

Jalali A, Ferguson M (1989) Computationally efficient control algorithms for Markov chains. In: Proceedings of the 28th conference on decision and control, Tampa, pp 1283–1288

Korf R (1990) Real-time heuristic search. Artif Intell 42(2–3):189–211

Smith T, Simmons R (2006) Focused real-time dynamic programming for MDPs: squeezing more out of a heuristic. In: Proceedings of the national conference on artificial intelligence (AAAI). AAAI Press, Boston

Sutton R (1990) Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: Proceedings of the 7th international conference on machine learning. Morgan Kaufmann, San Mateo, pp 216–224

Adaptive Resonance Theory

Gail A. Carpenter[1] and Stephen Grossberg[2]
[1]Department of Mathematics & Center for Adaptive Systems, Boston University, Boston, MA, USA
[2]Center for Adaptive Systems, Graduate Program in Cognitive and Neural Systems, Department of Mathematics, Boston University, Boston, MA, USA

Abstract

Computational models based on cognitive and neural systems are now deeply embedded in the standard repertoire of machine learning and data mining methods, with intelligent learning systems enhancing performance in nearly every existing application area. Beyond data mining, this article shows how models based on adaptive resonance theory (ART) may provide entirely new questions and practical solutions for technological applications. ART models carry out hypothesis testing, search, and incremental fast or slow, self-stabilizing learning, recognition, and prediction in response to large nonstationary databases (big data). Three computational examples, each based on the distributed ART neural network, frame questions and illustrate how a learning system (each with no free parameters) may enhance the analysis of large-scale data. Performance of each task is simulated on a common mapping platform, a remote sensing dataset called the Boston Testbed, available online along with open-source system code. Key design elements of ART models and links to software for each system are included. The article further points to future applications for integrative ART-based systems that have already been computationally specified and simulated. New application directions include autonomous robotics, general-purpose machine vision, audition, speech recognition, language acquisition, eye movement control, visual search, figure-ground separation, invariant object recognition, social cognition, object and spatial attention, scene understanding, space-time integration, episodic memory, navigation, object tracking, system-level analysis of mental disorders, and machine consciousness.

Adaptive Resonance Theory

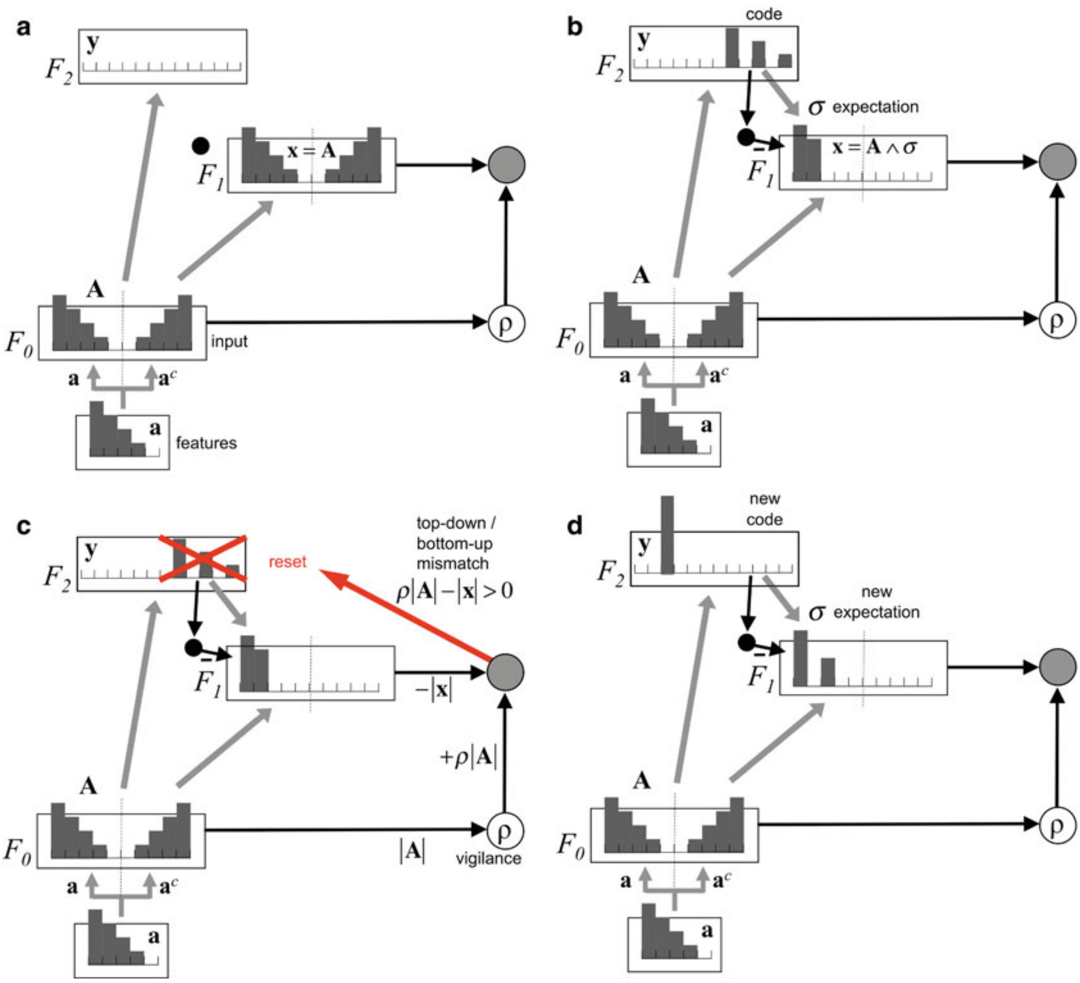
Adaptive resonance theory (ART) neural networks model real-time hypothesis testing, search, learning, recognition, and prediction. Since the 1980s, these models of human cognitive information processing have served as computational engines for a variety of neuromorphic technologies (http://techlab.bu.edu/resources/articles/C5). This article points to a broader range of technology transfers that bring new methods to new problem domains. It describes applications of three specific systems, ART knowledge discovery, self-supervised ART, and biased ART, and summarizes future application areas for large-scale, brain-based model systems.

ART Design Elements

In this article, *ART* refers generally to a theory of cognitive information processing and to an inclusive family of neural models. Design principles derived from scientific analyses and design constraints imposed by targeted applications have jointly guided the development of variants of the basic systems.

Stable Fast Learning with Distributed and Winner-Take-All Coding

ART systems permit fast online learning, whereby long-term memories reach their asymptotes on each input trial. With slow learning, memories change only slightly on each trial. One characteristic that distinguishes classes of ART systems from one another is the nature of their patterns of persistent activation at the coding field F_2 (Fig. 1). The coding field is functionally analogous to the hidden layer of multilayer perceptrons (*Encyclopedia* cross reference). At the perceptron hidden layer, activation is distributed across many nodes, learning needs



Adaptive Resonance Theory, Fig. 1 Distributed ART (dART) (Carpenter 1997). (**a**) At the field F_0, complement coding transforms the feature pattern **a** to the system input **A**, which represents both scaled feature values $a_i \in [0, 1]$ and their complements $(1 - a_i)$ $(i = 1 \ldots M)$. (**b**) F_2 is a competitive field that transforms its input pattern into the working memory code **y**. The F_2 nodes that remain active following competition send the pattern σ of learned top-down expectations to the match field F_1. The pattern active at F_1 becomes $\mathbf{x} = \mathbf{A} \wedge \sigma$, where $\wedge$ denotes the component-wise minimum, or fuzzy intersection. (**c**) A parameter $\rho \in [0, 1]$, called vigilance, sets the matching criterion. The system registers a mismatch if the size of **x** is less than ρ times the size of **A**. A top-down/bottom-up mismatch triggers a signal that resets the active F_2 code. (**d**) Medium-term memories in the F_0-to-F_2 dynamic weights allow the system to activate a new code **y**. When only one F_2 node remains active following competition, the code is maximally compressed, or winner-take-all. When $|\mathbf{x}| \geq \rho|\mathbf{A}|$, the activation pattern **y** persists until the next reset, even if input **A** changes or F_0-to-F_2 signals habituate. During learning, thresholds τ_{ij} in paths from F_0 to F_2 increase according to the dInstar law; and thresholds τ_{ji} in paths from F_2 to F_1 increase according to the dOutstar law

to be slow, and activation does not persist once inputs are removed. The ART coding field is a competitive network where, typically, one or a few nodes in the normalized F_2 pattern **y** sustain persistent activation, even as their generating inputs shift, habituate, or vanish. The pattern **y** persists until an active reset signal (Fig. 1c) prepares the coding field to register a new F_0-to-F_2 input. Early ART networks (Carpenter and Grossberg 1987; Carpenter et al. 1991a, 1992) employed *localist*, or *winner-take-all*, coding, whereby strongly competitive feedback results in only one F_2 node staying active until the next reset. With fast as well as slow learning, memory stability in these early networks relied on their winner-take-all architectures.

Achieving stable fast learning with distributed code representations presents a computational challenge to any learning network. In order to meet this challenge, distributed ART (Carpenter 1997) introduced a new network configuration (Fig. 1) in which system fields are identified with cortical layers (Carpenter 2001). New learning laws (*dInstar* and *dOutstar*) that realize stable fast learning with distributed coding predict adaptive dynamics between cortical layers.

Distributed ART (dART) systems employ a new unit of long-term memory, which replaces the traditional multiplicative weight (*Encyclopedia* cross reference) with a *dynamic weight* (Carpenter 1994). In a path from the F_2 coding node j to the F_1 matching node i, the dynamic weight equals the amount by which coding node activation y_j exceeds an *adaptive threshold* τ_{ji}. The total signal σ_i from F_2 to the i^{th} F_1 node is the sum of these dynamic weights, and F_1 node activation x_i equals the minimum of the top-down expectation σ_i and the bottom-up input A_i. During dOutstar learning, the top-down pattern σ converges toward the matched pattern $\mathbf{x}$.

When coding node activation $y_j$is below τ_{ji}, the dynamic weight is zero and no learning occurs in that path, even if y_j is positive. This property is critical for stable fast learning with distributed codes. Although the dInstar and dOutstar laws are compatible with F_2 patterns $\mathbf{y}$ that are arbitrarily distributed, in practice, following an initial learning phase, most changes in paths to and from a coding node j occur only when its activation y_j is large. This type of learning is therefore called *quasi-localist*. In the special case where coding is winner-take-all, the dynamic weight is equivalent to a multiplicative weight that formally equals the *complement* of the adaptive threshold.

Complement Coding: Learning Both Absent Features and Present Features

ART networks employ a preprocessing step called *complement coding* (Carpenter et al. 1991b), which models the nervous system's ubiquitous computational design known as *opponent processing* (Hurvich and Jameson 1957). Balancing an entity against its opponent, as in opponent colors such as red vs. green or agonist-antagonist muscle pairs, allows a system to act upon relative quantities, even as absolute magnitudes fluctuate unpredictably. In ART systems, complement coding is analogous to retinal on-cells and off-cells (Schiller 1982). When the learning system is presented with a set of input features $\mathbf{a} \equiv (a_1 \ldots a_i \ldots a_M)$, complement coding doubles the number of input components, presenting to the network an input $\mathbf{A}$ that concatenates the original feature vector and its complement (Fig. 1a).

Complement coding produces normalized inputs $\mathbf{A}$ that allow a model to encode features that are consistently *absent* on an equal basis with features that are consistently *present*. Features that are sometimes absent and sometimes present when a given F_2 node is highly active are regarded as uninformative with respect to that node, and the corresponding *present* and *absent* top-down feature expectations shrink to zero. When a new input activates this node, these features are suppressed at the match field F_1 (Fig. 1b). If the active code then produces an error signal, attentional biasing can enhance the salience of input features that it had previously ignored, as described below.

Matching, Attention, and Search

A neural computation central to both scientific and technological analyses is the *ART matching rule* (Carpenter and Grossberg 1987), which controls how attention is focused on critical feature patterns via dynamic matching of a bottom-up sensory input with a top-down learned expectation. Bottom-up/top-down pattern matching and attentional focusing are, perhaps, the primary features common to all ART models across their many variations. Active input features that are not confirmed by top-down expectations are inhibited (Fig. 1b). The remaining activation pattern defines a focus of attention, which, in turn, determines what feature patterns are learned. Basing memories on attended features rather than whole patterns supports the design goal of encoding stable memories with fast as well as slow learning. Encoding attended feature subsets also enables one-to-many learning, where the system may attach many context-dependent labels (*Spot*, *dog*, *animal*) to one input. This capability promotes knowledge discovery ($Spot \Rightarrow dog$ and $dog \Rightarrow animal$) in a learning system that experiences one input at a time, with no explicit connection between inputs.

When the match is good enough, F_2 activation persists and learning proceeds. Where they exceed the corresponding bottom-up input components, top-down signals decay as expectations converge toward the attended pattern at F_1. The coding field F_2 contains a reserve of *uncommitted* coding nodes, which compete with the previously active *committed* nodes. When a previously uncommitted node is first activated during supervised learning, it is associated with its designated output class. During testing, the selection of an uncommitted node means *I don't know*. ART networks for supervised learning are called *ARTMAP* (Carpenter et al. 1991a, 1992).

A mismatch between an active top-down expectation and the bottom-up input leads to a parallel memory search (Fig. 1c). The ART matching criterion is set by a *vigilance parameter* ρ. Low vigilance permits the learning of broad classes, across diverse exemplars, while high vigilance limits learning to narrow classes. When a new input arrives, vigilance equals a baseline level. Baseline vigilance is set equal to zero to maximize generalization. ARTMAP vigilance increases following a predictive error or negative reinforcement (*Encyclopedia* cross reference). The internal computation that determines how far ρ rises to correct the error is called *match tracking* (Carpenter et al. 1991a). As vigilance rises, the network pays more attention to how well top-down expectations match the bottom-up input. The match tracking modification MT– (Carpenter and Markuzon 1998) also allows the system to learn inconsistent cases. For example, three similar, even identical, map regions may have been correctly labeled by different observers as *ocean* or *water* or *natural*. The ability to learn one-to-many maps, which can label a single test input as *ocean* and *water* and *natural*, is a key feature of the ART knowledge discovery system described below.

Applications

Three computational examples illustrate how cognitive and neural systems can introduce new approaches to the analysis of large datasets. Application 1 (self-supervised ART) addresses the question: how can a neural system learning from one example at a time absorb information that is inconsistent but correct, as when a family pet is called *Spot* and *dog* and *animal*, while rejecting similar incorrect information, as when the same pet is called *wolf*? How does this system transform scattered information into knowledge that *dogs are animals*, but not conversely? Application 2 (ART knowledge discovery) asks: how can a real-time system, initially trained with a few labeled examples and a limited feature set, continue to learn from experience, without supervision, when confronted with oceans of additional information, without eroding reliable early memories? How can such individual systems adapt to their unique application contexts? Application 3 (biased ART) asks: how can a neural system that has made an error refocus attention on features that it initially ignored?

The Boston Testbed

The Boston Testbed was developed to compare performance of learning systems applied to challenging problems of spatial analysis. Each multispectral Boston image pixel produces 41 feature values: 6 Landsat 7 Thematic Mapper (TM) bands at 30 m resolution, 2 thermal bands at 60 m resolution, 1 panchromatic band at 15 m resolution, and 32 derived bands representing local contrast, color, and texture. In the Boston dataset, each of 28,735 ground truth pixels is labeled as belonging to one of seven classes (*beach, ocean, ice, river, park, residential, industrial*). For knowledge discovery system training, some *ocean, ice*, and *river* pixels are instead labeled as belonging to broader classes such as *water* or *natural*. No pixel has more than one label, and the learning system is given no information about relationships between target classes. The labeled dataset is available from the CNS Technology Lab Website [http://techlab.bu.edu/classer/data_sets/].

A cross-validation procedure divides an image into four vertical strips: two for training, one for validation (if needed for parameter selection), and one for testing. Class mixtures differ markedly across strips. For example, one strip contains many *ocean* pixels, while another strip contains neither *ocean* nor *beach* pixels. Geographically dissimilar training and testing areas robustly assess regional generalization. In this article, spatial analysis simulations on the Boston Testbed follow this protocol to illustrate ART systems for self-supervised learning, knowledge discovery, and attentional control. Since each system in Applications 1–3 requires no parameter selection, training uses randomly chosen pixels from three strips, with testing on the fourth strip.

Application 1: Learning from Experience with Self-Supervised ART

Computational models of supervised pattern recognition typically utilize two learning phases. During an initial training phase, input patterns, described as specified values of a set of features, are presented along with output class labels or patterns. During a subsequent testing phase, the model generates output predictions for unlabeled inputs, and no further learning takes place.

Although supervised learning has been successfully applied in diverse settings, it does not reflect many natural learning situations. Humans do learn from explicit training, as from a textbook or a teacher, and they do take tests. However, students do not stop learning when they leave the classroom. Rather, they continue to learn from experience, incorporating not only more information but new types of information, all the while building on the foundation of their earlier knowledge. Self-supervised ART models such life-long learning.

An unsupervised learning system clusters unlabeled input patterns. *Semi-supervised* learning incorporates both labeled and unlabeled inputs in its training set, but all inputs typically have the same number of specified feature values. Without any novel features from which to learn, semi-supervised learning systems use unlabeled data to refine the model parameters defined using labeled data. Reviews of semi-supervised learning (Chapelle et al. 2006) have found that many of the successful models are carefully selected and tuned, using a priori knowledge of the problem. Chapelle et al. (2006) conclude that none of the semi-supervised models they review is robust enough to be general purpose. The main difficulty seems to be that, whenever unlabeled instances are different enough from labeled instances to merit learning, these differences could contain misinformation that may damage system performance.

The *self-supervised* paradigm models two learning stages. During Stage 1 learning, the system receives all output labels, but only a subset of possible feature values for each input. During Stage 2 learning, the system may receive more feature values for each input, but no output labels. In Stage 1, when the system can confidently incorporate externally specified output labels, self-supervised ART (Amis and Carpenter 2010) employs winner-take-all coding and fast learning. In Stage 2, when the system internally generates its own output labels, codes are distributed so that incorrect hypotheses do not abruptly override reliable "classroom learning" of Stage 1. The distributed ART learning laws, dInstar (Carpenter 1997) and dOutstar (Carpenter 1994), scale memory changes to internally generated measures of prediction confidence and prevent memory changes altogether for most inputs. Memory stability derives from the dynamic weight representation of long-term memories, which permits learning only in paths to and from highly active coding nodes. Dynamic weights solve a problem inherent in learning laws based on multiplicative weights, which are prone to catastrophic forgetting when implemented with distributed codes and huge datasets, even when learning is very slow.

In addition to emulating the human learning experience, self-supervised learning maps to technological applications that need to cope with huge, ever-changing datasets. A supervised learning system that completes all training before making test predictions does not adapt to new information and individual contexts. A semi-supervised system risks degrading its supervised knowledge. Self-supervised ART continues to learn from new experiences, with built-in safeguards that conserve useful memories. Self-supervised ART code is available from the CNS Technology Lab Website (http://techlab.bu.edu/SSART/).

A simulation study based on the Boston Testbed (Amis and Carpenter 2010) illustrates ways in which high-dimensional problems may challenge *any system* learning without labels. As in most ground truth datasets, labeled pixels consist primarily of clear exemplars of single classes. Because sensors have a 15–60 m resolution, many unlabeled pixels cover multiple classes, such as *ice* and *industrial*. Stage 2 inputs thus mix and distort features from multiple classes, placing many of the unlabeled feature vectors far from the distinct class clusters of the Stage 1 training set. Although the distributed ART learning laws are open to unrestricted adaptation on any pixel, the distributed codes of Stage 2 minimize the influence of mixed pixels. Most memory changes occur on unambiguous cases, despite the fact that the unlabeled pixels provide no external indices of class ambiguity. Self-supervised Stage 2 learning dramatically improves performance compared to learning that ends after Stage 1. On every one of 500 individual simulations, Stage 2 learning improves test accuracy, as unlabeled fully featured inputs consistently expand knowledge from Stage 1 training.

Application 2: Transforming Information into Knowledge Using ART Knowledge Discovery

Classifying terrain or objects may require the resolution of conflicting information from sensors working at different times, locations, and scales and from users with different goals and situations. *Image fusion* has been defined as "the acquisition, processing and synergistic combination of information provided by various sensors or by the same sensor in many measuring contexts" (Simone et al. 2002, p. 3). When multiple sources provide inconsistent data, fusion methods are called upon to appraise information components to decide among various options and to resolve inconsistencies, as when evidence suggests that an object is a *car* or a *truck* or a *bus*. Fusion methods weigh the confidence and reliability of each source, merging complementary information or gathering more data. In any case, at most one of these answers is correct.

The method described here defines a complementary approach to the information fusion problem, considering the case where sensors and sources are both nominally inconsistent and reliable, as when evidence suggests that an object is a *car* and a *vehicle* and *man-made* or when a *car* is alternatively labeled *automobile*. Underlying relationships among classes are assumed to be unknown to the automated system or the human user, as if the labels were encrypted.

The ART knowledge discovery model acts as a self-organizing expert system to derive consistent knowledge structures from such nominally inconsistent data (Carpenter et al. 2005). Once derived, a rule set can be used to assign classes to levels. For each rule $x \Rightarrow y$, class x is located at a lower level than class$y$. Classes connected by arrows that codify a list of rules and confidence values form a graphical representation of a knowledge hierarchy. For spatial data, the resulting diagram of the relationships among classes can guide the construction of orderly layered maps. ART knowledge discovery code is available from the CNS Technology Lab Website (http://techlab.bu.edu/classer/classer_toolkit_overview). On the Boston Testbed, the ART knowledge discovery system places each class at its correct level and finds all the correct rules for this example.

Application 3: Correcting Errors by Biasing Attention Using Biased ART

Memories in ART networks are based on matched patterns that focus attention on *critical features*, where bottom-up inputs match active top-down expectations. While this learning strategy has proved successful for both brain models and applications, computational examples demonstrate that paying too much attention to critical features that have been selected to represent a given category early on may distort memory representations during subsequent learning. If training inputs are repeatedly presented, an ART system will correct these initial errors. However, real-time learning may not afford such repeat opportunities. Biased ART (bART) (Carpenter and Gaddam 2010) solves the problem of overemphasis on early critical features by directing attention away from initially attended features after the system makes a predictive error.

Activity $\mathbf{x}$ at the ART field F_1 computes the match between the field's bottom-up and top-down input patterns (Fig. 1). A reset signal shuts off the active F_2 code when $\mathbf{x}$ fails to meet the matching criterion determined by vigilance ρ. Reset alone does not, however, induce a different code: unless the prior code has left an enduring trace within the F_0–F_2 subsystem, the network will simply reactivate the same pattern at F_2.

Following reset, all ART systems shift attention away from previously active *coding* nodes at the field F_2. As modeled in ART 3 (Carpenter and Grossberg 1990), biasing the bottom-up input to the coding field to favor previously inactive F_2 nodes implements search by enabling the network to activate a new code in response to a reset signal. The ART 3 search mechanism defines a medium-term memory in the F_0-to-F_2 adaptive filter so that the system does not perseverate indefinitely on an output class that had just produced a reset. A presynaptic interpretation of this bias mechanism is transmitter depletion or habituation.

The biased ART network (Carpenter and Gaddam 2010) introduces a second, top-down, medium-term memory which, following reset, shifts attention away from previously active *feature* nodes at the match field F_1. In Fig. 1, the first feature is strongly represented in the input $\mathbf{A}$ and in the matched patterns $\mathbf{x}$ at F_1 both before reset (Fig. 1b) and after reset (Fig. 1d). Following the same sequence as in Fig. 1a–c, biased ART would diminish the size of the first feature in the matched pattern. The addition of featural biasing helps the system to pay more attention to input features that it had previously ignored.

The biasing mechanism is a small modular element that can be added to any ART network. While computational examples and Boston Testbed simulations demonstrate how featural biasing in response to predictive errors improves performance on supervised learning tasks, the error signal that gates biasing could have originated from other sources, as in reinforcement learning. Biased ART code is available from the CNS Technology Lab Website (http://techlab.bu.edu/bART).

Future Directions

Applications for tested software based on computational intelligence abound. This section outlines areas where ART systems may open qualitatively new frontiers for novel technologies. Future applications summarized here would adapt and specialize brain models that have already been mathematically specified and computationally simulated to explain and predict large psychological and neurobiological databases. By linking the brain to mind, these models characterize both mechanism (how the model works) and function (what the model is for). Both mechanism and function are needed to design new applications. These systems embody new designs for autonomous adaptive agents, including new computational paradigms that are called Complementary Computing and Laminar Computing. These paradigms enable the autonomous adaptation in real time of individual persons or machines to nonstationary situations filled with unexpected events. See Grossberg (2013) for a review.

New Paradigms for Autonomous Intelligent Systems: Complementary Computing and Laminar Computing

Functional integration is essential to the design of a complex autonomous system such as a robot moving and learning freely in an unpredictable environment. Linking independent modules for, say, vision and motor control will not necessarily produce a coordinated system that can adapt to unexpected events in changeable contexts. How, then, should such an autonomous adaptive system be designed?

A clue can be found in the nature of brain specialization. How have brains evolved while interacting with the physical world and embodying its invariants? Many scientists have proposed that our brains possess independent modules. The brain's organization into distinct anatomical areas and processing streams shows that brain regions are indeed specialized. Whereas independent modules compute their particular processes on their own, behavioral data argue against this possibility. *Complementary Computing* (Grossberg 2000a,b, 2013) concerns the discovery that pairs of parallel cortical processing streams compute computationally complementary properties. Each stream has complementary strengths and weaknesses, much as in physical principles like the Heisenberg uncertainty principle. Each cortical stream can also possess multiple processing stages. These stages realize a *hierarchical resolution of uncertainty*. "Uncertainty" here means that computing one set of properties at a given stage prevents computation of a complementary set of properties at that stage. Complementary Computing proposes that the computational unit of brain processing that has behavioral significance consists of parallel and hierarchical interactions between complementary cortical processing streams with multiple processing stages. These interactions overcome complementary weaknesses to compute necessary information about a particular type of biological intelligence.

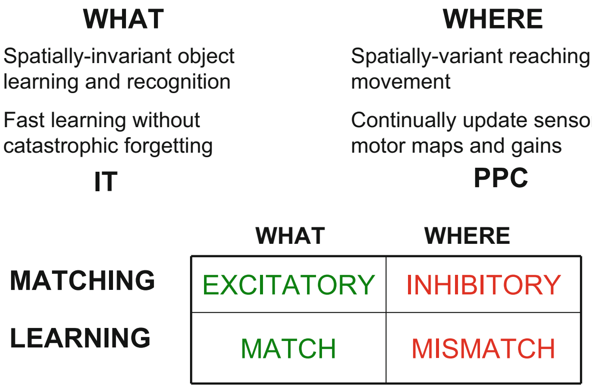
Five decades of neural modeling have shown how Complementary Computing is embedded as a fundamental design principle in neural systems for vision, speech and language, cognition, emotion, and sensory-motor control. Complementary Computing hereby provides a blueprint for designing large-scale autonomous adaptive systems that are poised for technological implementation.

A unifying anatomical theme that enables communication among cortical systems is *Laminar Computing*. The cerebral cortex, the seat of higher intelligence in all modalities, is organized into layered circuits (often six main layers) that undergo characteristic bottom-up, top-down, and horizontal interactions. As information travels up and down connected regions, distributed decisions are made in real time based on a preponderance of evidence. Multiple levels suppress weaker groupings while communicating locally coherent choices. The distributed ART model (Fig. 1), for example, features three cortical layers, with its distributed code (e.g., at a cortical layer 6) producing a distributed output. Stacks of match fields (inflow) and coding fields (outflow) lay the substrate for cortical hierarchies.

How do specializations of this shared laminar design embody different types of biological intelligence, including vision, speech, language, and cognition? How does this shared design enable seamless intercortical interactions? Models of Laminar Computing clarify how these different types of intelligence all use variations of the same laminar circuitry (Grossberg 2013; Grossberg and Pearson 2008). This circuitry represents a revolutionary synthesis of desirable computational properties of feedforward and feedback processing, digital and analog processing, and bottom-up data-driven processing and top-down attentive hypothesis-driven processing. Realizing such designs in hardware that embodies biological intelligence promises to facilitate the development of increasingly general-purpose adaptive autonomous systems for multiple applications.

Complementary Computing in the Design of Perceptual/Cognitive and Spatial/Motor Systems

Many neural models that embody subsystems of an autonomous adaptive agent have been developed and computationally characterized. It remains to unify and adapt them to particular machine learning applications. Complementary Computing implies that not all of these subsystems could be based on variants of ART. In particular, accumulating experimental and theoretical evidence shows that perceptual/cognitive and spatial/motor processes use different learning, matching, and predictive laws for their complementary functions (Fig. 2). ART-like processing is ubiquitous in perceptual and cognitive processes, including excitatory matching and match-based learning that enables self-stabilizing memories to form. Vector


WHAT
Spatially-invariant object learning and recognition
Fast learning without catastrophic forgetting
IT
WHERE
Spatially-variant reaching and movement
Continually update sensory-motor maps and gains
PPC

	WHAT	WHERE
MATCHING	EXCITATORY	INHIBITORY
LEARNING	MATCH	MISMATCH

Adaptive Resonance Theory, Fig. 2 Complementary What and Where cortical processing streams for spatially invariant object recognition and spatially variant spatial representation and action, respectively. Perception and recognition use top-down excitatory matching and match-based fast or slow learning without catastrophic forgetting. Spatial and motor tasks use inhibitory matching and mismatch-based learning to achieve adaptation to changing bodily parameters. *IT* inferotemporal cortex, *PPC* posterior parietal cortex

Associative Map (VAM) processing is often found in spatial and motor processes, which rely on inhibitory matching and mismatch-based learning. In these modalities, spatial maps and motor plants are adaptively updated without needing to remember past maps and parameters. Complementary mechanisms create a self-stabilizing perceptual/cognitive front end for intelligently manipulating the more labile spatial/motor processes that enable our changeable bodies to act effectively upon a changing world.

Some of the existing large-scale ART systems are briefly reviewed here, using visually based systems for definiteness. Citations refer to articles that specify system equations and simulations and that can be downloaded from http://cns.bu.edu/~steve.

Where's Waldo? Unifying Spatial and Object Attention, Learning, Recognition, and Search of Valued Objects and Scenes

ART models have been incorporated into larger system architectures that clarify how individuals autonomously carry out intelligent behaviors as they explore novel environments. One such development is the ARTSCAN family of architectures, which model how individuals rapidly learn to search a scene to detect, attend, invariantly recognize, and look at a valued target object (Fig. 3; Cao, Grossberg, and Markowitz 2011; Chang, Grossberg, and Cao 2014; Fazl, Grossberg, and Mingolla 2009; Foley, Grossberg, and Mingolla 2012; Grossberg, Srinivasan, and Yazdanbakhsh 2014). Such a competence represents a proposed solution of the Where's Waldo problem.

The ventral What stream is associated with object learning, recognition, and prediction, whereas the dorsal Where stream carries out processes such as object localization, spatial attention, and eye movement control. To achieve efficient object recognition, the What stream learns object category representations that become increasingly invariant under view, size, and position changes at higher processing stages. Such invariance enables objects to be learned and recognized without causing a combinatorial explosion. However, by stripping away the positional coordinates of each object exemplar, the What stream loses the ability to command actions to the positions of valued objects. The Where stream computes positional representations of the world and controls actions to acquire objects in it, but does not represent detailed properties of the objects themselves.

ARTSCAN architectures model how an autonomous agent can determine when the views that are foveated by successive scanning movements belong to the same object and thus determine which view-selective categories should be associatively linked to an emerging view- , size-, and positionally-invariant object category. This competence, which avoids the problem of erroneously merging pieces of different objects, works even under the unsupervised learning conditions that are the norm during many object learning experiences in vivo. The model identifies a new role for spatial attention in the Where stream, namely, control of invariant object category learning by the What stream. Interactions across the What and Where streams overcome the deficiencies of computationally complementary properties of these streams.

In the ARTSCAN Search model, both Where-to-What and What-to-Where stream interactions are needed to overcome complementary weaknesses: Where stream processes of spatial attention and predictive eye movement control regulate What stream processes whereby multiple view- and positionally-specific object categories are learned and associatively linked to view- and positionally-invariant object categories through bottom-up and object-attentive top-down interactions. What stream cognitive-emotional learning processes enable the focusing of motivated attention upon the invariant object categories of desired objects (Brown, Bullock, and Grossberg 1999, 2004; Dranias, Grossberg, and Bullock 2008; Grossberg and Seidman 2006). What stream cognitive names or motivational drives can, together with volitional signals, drive a search for Waldo. Mediated by object attention, search proceeds from What stream positionally-invariant representations to Where stream positionally-specific representations that focus spatial attention on Waldo's position. ARTSCAN architectures hereby model how the dynamics of multiple brain regions are coordinated to achieve clear functional goals.

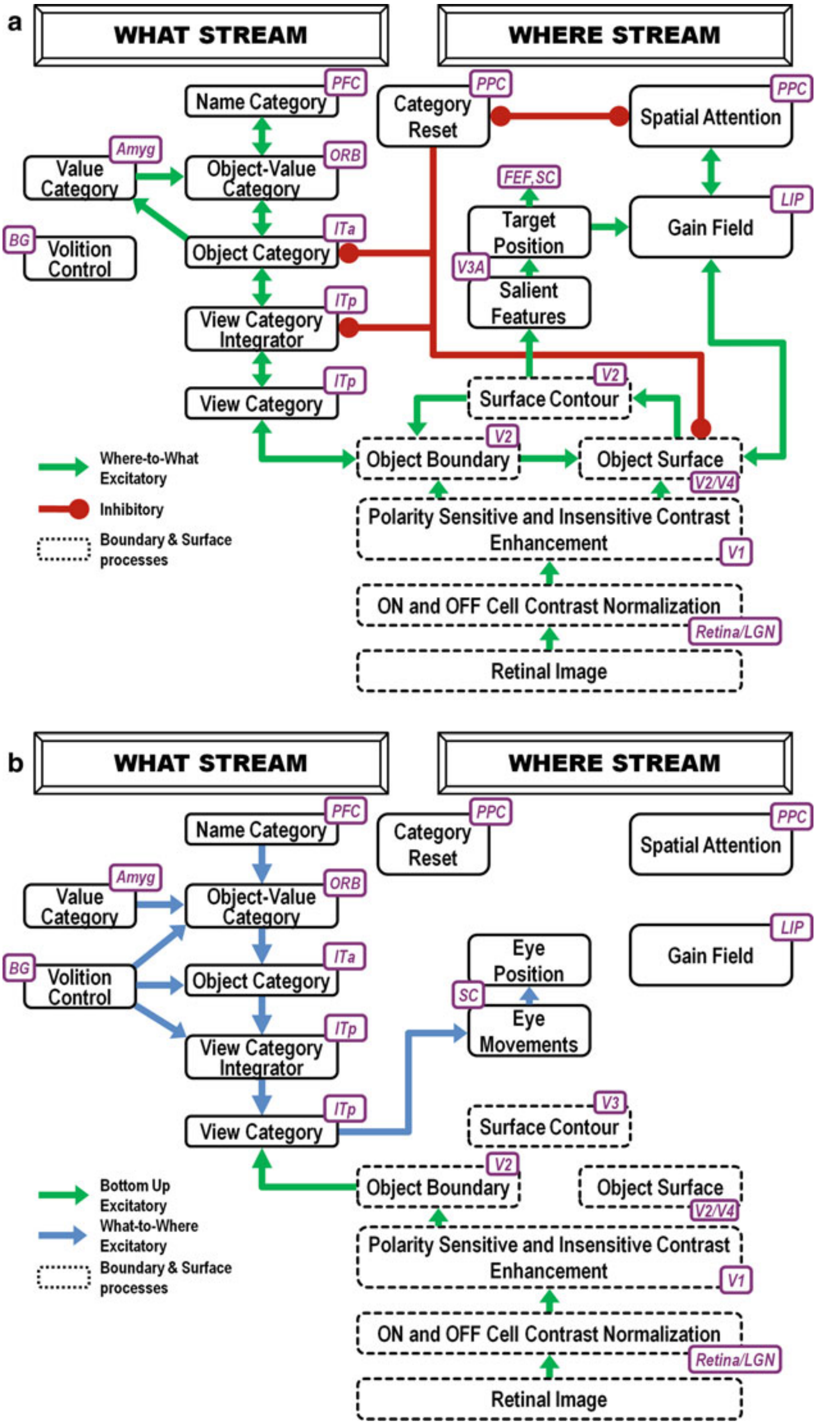
The focus of spatial attention on Waldo's position in the Where stream can be used to control eye and hand movements toward Waldo, after navigational circuits (see below) bring the observer close enough to contact him. VAM-type learning circuits have been developed for the control of goal-oriented eye and hand movements that can be used for this purpose (e.g., Bullock and Grossberg 1988, 1991; Bullock, Cisek, and Grossberg 1998; Contreras-Vidal, Grossberg, and Bullock 1997; Gancarz and Grossberg 1999; Grossberg, Srihasam, and Bullock 2012; Pack, Grossberg, and Mingolla 2001; Srihasam, Bullock, and Grossberg 2009).

The ARTSCENE system (Grossberg and Huang 2009) models how humans can incrementally learn and rapidly predict scene identity by gist and then accumulates learned evidence from scenic textures to refine its initial hypothesis, using the same kind of spatial attentional shrouds that help to learn invariant object categories in ARTSCAN. The ARTSCENE Search system (Huang and Grossberg 2010) models how humans use target-predictive contextual information to guide search for desired targets in familiar scenes. For example, humans can learn that a certain combination of objects may define a context for a kitchen and trigger a more efficient search for a typical object, such as a sink, in that context.

General-Purpose Vision and How It Supports Object Learning, Recognition, and Tracking

Visual preprocessing constrains the quality of visually based learning and recognition. On an assembly line, automated vision systems successfully scan for target objects in this carefully controlled environment. In contrast, a human or robot navigating a natural scene faces overlaid textures, edges, shading, and depth information, with multiple scales and shifting perspectives. In the human brain, evolution has produced a huge preprocessor, involving multiple brain regions, for object and scene representation and for target tracking and navigation. One reason for this is that visual boundaries and surfaces, visual form and motion, and target tracking and visually based navigation are computationally complementary, thus requiring several distinct but interacting cortical processing streams.

Prior to the development of systems such as ARTSCAN and ARTSCENE, the FACADE (Form-And-Color-And-DEpth) model provided



Adaptive Resonance Theory, Fig. 3 (continued) a neural theory of form perception, including 3D vision and figure-ground separation (e.g., Cao and Grossberg 2005, 2012; Fang and Grossberg 2009; Grossberg, Kuhlmann, and Mingolla 2007; Grossberg and Swaminathan 2004; Kelly and Grossberg 2000). The 3D FORMOTION model provides a neural theory of motion processing and form-motion interactions (e.g., Baloch and Grossberg 1997; Baloch, Grossberg, Mingolla, and Nogueira 1999; Berzhanskaya, Grossberg, and Mingolla 2007; Grossberg, Leveille, and Versace 2011; Grossberg, Mingolla, and Viswanathan 2001; Grossberg and Rudd 1992). The FACADE model has just the properties that are needed for solving the Where's Waldo problem, and the 3D FORMOTION model has just the properties that are needed for tracking unpredictably moving targets. Their complementary properties enabled these extensions.

Visual and Spatial Navigation, Cognitive Working Memory, and Planning

In addition to being able to see, learn, recognize, and track valued goal objects, an animal or autonomous robot must also be able to navigate to or away from them and to interact with them through goal-oriented hand and arm movements. Navigation is controlled by two distinct and interacting systems: a visually guided system and a spatial path integration system.

Visually guided navigation through a cluttered natural scene is modeled using the 3D FORMOTION model as a front end. The STARS and ViSTARS neural systems (Browning, Grossberg, and Mingolla 2009a,b; Elder, Grossberg, and Mingolla 2009) model how primates use object motion information to segment objects and optic flow information to determine heading (self-motion direction), for purposes of goal approach and obstacle avoidance in response to realistic environments. The models predict how computationally complementary processes in parallel streams within the visual cortex compute object motion for tracking and self-motion for navigation. The models' steering decisions compute goals as attractors and obstacles as repellers, as do humans.

Spatial navigation based upon path integration signals has been a topic of great interest recently. Indeed, the 2014 Nobel Prize in Physiology or Medicine was awarded to John O'Keefe for his discovery of place cells in the hippocampal cortex and to Edvard and May-Britt Moser for their discovery of grid cells in the entorhinal cortex. The GridPlaceMap neural system (Grossberg and Pilly 2012, 2014; Pilly and Grossberg 2012, 2014; Mhatre, Grossberg, and Gorchetch-

Adaptive Resonance Theory, Fig. 3 ARTSCAN Search macrocircuit and corresponding brain regions. *Dashed boxes* indicate boundary and surface preprocessing. (**a**) Category learning system. *Arrows* represent excitatory cortical processes. Spatial attention in the Where stream regulates view-specific and view-invariant category learning and recognition, and attendant reinforcement learning, in the What stream. Connections ending in circular disks indicate inhibitory connections. (**b**) Where's Waldo search system. Search begins when a name category or value category is activated and subliminally primes an object-value category via the *ART matching rule*. A volition control signal enables the primed object-value category to fire output signals. Bolstered by volitional control signals, these output signals can, in turn, propagate through a positionally-invariant object category to all the positionally-variant view category integrators whose various views and positions are represented by the object category. The view category integrators can subliminally prime, but not fully activate, these view categories. All this occurs in the What stream. When the bottom-up input from an object's boundary/surface representation also activates one of these view categories, its activity becomes suprathreshold, wins the competition across view categories for persistent activation, and activates a spatial attentional representation of Waldo's position in the Where stream. *ITa* anterior part of inferotemporal cortex, *ITp* posterior part of inferotemporal cortex, *PPC* posterior parietal cortex, *LIP* lateral intraparietal cortex, *LGN* lateral geniculate nucleus, *ORB* orbitofrontal cortex, *Amyg* amygdala, *BG* basal ganglia, *PFC* prefrontal cortex, *SC* superior colliculus, *V1* striate visual cortex, *V2, V3,* and *V4* prestriate visual cortices nikov 2012; Pilly and Grossberg 2014) proposes how entorhinal grid cells and hippocampal place cells may be learned as spatial categories in a hierarchy of self-organizing maps. The model responds to realistic rat navigational trajectories by learning both grid cells with hexagonal grid firing fields of multiple spatial scales, and place cells with one or more firing fields. Model dynamics match neurophysiological data about their development in juvenile rats. The GridPlaceMap model enjoys several parsimonious design features that will facilitate their embodiment in technological applications, including hardware: (1) similar ring attractor mechanisms process both linear and angular path integration inputs that drive map learning; (2) the same self-organizing map mechanisms can learn grid cell and place cell receptive fields in a hierarchy of maps, and both grid and place cells can develop by detecting, learning, and remembering the most frequent and energetic co-occurrences of their inputs; and (3) the learning of the dorsoventral organization of grid cell modules with multiple spatial scales that occur in the pathway from the medial entorhinal cortex to hippocampus seems to use mechanisms that are homologous to those for adaptively timed temporal learning that occur in the pathway from the lateral entorhinal cortex to hippocampus (Grossberg and Merrill 1989, 1992; Grossberg and Schmajuk 1989). The homologous mechanisms for representing space and time in this entorhinal-hippocampal system has led to the phrase “neural relativity” for this parsimonious design.

Finally, the GridPlaceMap model is an ART system. It proposes how top-down hippocampus-to-entorhinal attentional mechanisms may stabilize map learning and thereby simulates how hippocampal inactivation may disrupt grid cell properties and explains challenging data about theta, beta, and gamma oscillations.

Visual and path integration information cooperate during navigation. Cognitive planning also influences navigational decisions. More research is needed to show how learning fuses visual, path integration, and planning circuits into a unified navigational system. The design of a general planning system will be facilitated by the fact that similar circuits for short-term storage of event sequences (working memory) and for learning of sequential plans are used by the brain to control linguistic, spatial, and motor behaviors (Grossberg and Pearson 2008; Silver, Grossberg, Bullock, Histed, and Miller 2011).

Social Cognition

How can multiple autonomous systems interact intelligently? Individuals experience the world from self-centered perspectives. What we learn from each other is thus computed in different coordinates within our minds. How do we bridge these diverse coordinates? A model of social cognition that explains how a teacher can instruct a learner who experiences the world from a different perspective can be used to enable a single human or robotic teacher to instruct a large “class” of embodied robots that all experience the teacher from different perspectives.

Piaget’s *circular reaction* notes the feedback loop between the eye and hand in the learning infant, laying the foundation for visually guided reaching. Similarly, feedback between babbled sounds and hearing forms the learned substrate of language production. These *intra*personal circular reactions were extended to *inter*personal circular reactions within the Circular Reactions for Imitative Behavior (CRIB) model (Grossberg and Vladusich 2010). This model shows how social cognition builds upon ARTSCAN mechanisms. These mechanisms clarify how an infant learns how to share joint attention with adult teachers and to follow their gaze toward valued goal objects. The infant also needs to be capable of view-invariant object learning and recognition whereby it can carry out goal-directed behaviors, such as the use of tools, using different object views than the ones that its teachers use. Such capabilities are often attributed to *mirror neurons*. This attribution does not, however, explain the brain processes whereby these competences arise. CRIB proposes how intrapersonal circular reactions create a foundation for interpersonal circular reactions when infants and other learners interact with external teachers in space. Both types of circular reactions involve learned coordinate transformations between body-centered arm movement commands and retinotopic visual feedback, and coordination of processes within and between the What and Where cortical processing streams. Specific breakdowns of model processes generate formal symptoms similar to clinical symptoms of autism.

Mental Disorders and Homeostatic Plasticity

Optimally functioning autonomous intelligent systems require properly balanced complementary systems. What happens when they become imbalanced? In humans, they can experience mental disorders.

Scientific literature on human mental disorders such as autism and schizophrenia is, of necessity, more anecdotal than parametric and is, therefore, an insufficient foundation for model construction. Real-time models of normal mental behavior that are based on the huge databases from decades of psychological and neurobiological experiments have, however, provided insights into the mechanisms of abnormal behaviors (e.g., Carpenter and Grossberg 1993; Grossberg 1984, 2000a,b; Grossberg and Seidman 2006).

Imbalanced processes across the complementary systems that control normal behaviors can produce constellations of model symptoms that strikingly resemble mental disorders. For example, fixing the ART vigilance parameter ρ at too high a level leads to symptoms familiar in autistic individuals, notably learning of hyperconcrete categories and difficulty paying attention to the meaning of a task. Underarousal of the model amygdala can lead to insensitivity to social meanings and also to intense emotional outbursts and coping strategies to reduce event complexity and unexpectedness. Damage to the model cerebellum can lead to defects of adaptively timed learning and thus a host of problems in socialization.

In both humans and robots, it remains an open problem to model how biologically based autonomous systems can discover and maintain their own optimal operating parameters in response to the challenges of an unpredictable world. An initial step toward solving this *homeostatic plasticity* problem was made in Chandler and Grossberg (2012).

Machine Consciousness?

An early ART prediction is that *all conscious states are resonant states,* though not all resonant states are conscious. Since that time, ART has predicted how specific resonances support different kinds of consciousness. These observations suggest the question: can machines that embody ART resonant dynamics experience a type of consciousness? For example, ART models predict that *surface-shroud resonances* subserve conscious percepts of visual qualia, *feature-category resonances* subserve recognition of familiar objects and scenes, *spectral-shroud resonances* subserve conscious percepts of auditory streams, spectral-pitch-and-timbe resonances subserve conscious recognition of auditory streams, *item-list resonances* subserve conscious percepts of speech and language, and *cognitive-emotional resonances* subserve conscious feelings and knowing the objects or events that cause them. ART models also identify the brain regions and interactions that would support these resonances.

These results about model correlates of consciousness emerge from ART analyses of the mechanistic relationships among processes of Consciousness, Learning, Expectation, Attention, Resonance, and Synchrony (the CLEARS processes). Recall, however, that not all resonant states are conscious states. For example, entorhinal-hippocampal resonances are predicted to dynamically stabilize the learning of entorhinal grid cells and hippocampal place cells, and parietal-prefrontal resonances are predicted to trigger the selective opening of basal ganglia gates to enable the read-out of context-appropriate actions. Grossberg (2013; 2016) reviews these and other aspects of ART as a cognitive and neural theory.

Recommended Reading

Amis GP, Carpenter GA (2010) Self-supervised ARTMAP. Neural Netw 23:265–282

Baloch AA, Grossberg S (1997) A neural model of high-level motion processing: line motion and formotion dynamics. Vis Res 37:3037–3059

Baloch AA, Grossberg S, Mingolla E, Nogueira CAM (1999) A neural model of first-order and second-order motion perception and magnocellular dynamics. J Opt Soc Am A 16:953–978

Berzhanskaya J, Grossberg S, Mingolla E (2007) Laminar cortical dynamics of visual form and motion interactions during coherent object motion perception. Spat Vis 20:337–395

Brown J, Bullock D, Grossberg S (1999) How the basal ganglia use parallel excitatory and inhibitory learning pathways to selectively respond to unexpected rewarding cues. J Neurosci 19:10502–10511

Brown JW, Bullock D, Grossberg S (2004) How laminar frontal cortex and basal ganglia circuits interact to control planned and reactive saccades. Neural Netw 17:471–510

Browning A, Grossberg S, Mingolla M (2009a) A neural model of how the brain computes heading from optic flow in realistic scenes. Cogn Psychol 59:320–356

Browning A, Grossberg S, Mingolla M (2009b) Cortical dynamics of navigation and steering in natural scenes: motion-based object segmentation, heading, and obstacle avoidance. Neural Netw 22:1383–1398

Bullock D, Grossberg S (1988) Neural dynamics of planned arm movements: emergent invariants and speed-accuracy properties during trajectory formation. Psychol Rev 95:49–90

Bullock D, Grossberg S (1991) Adaptive neural networks for control of movement trajectories invariant under speed and force rescaling. Hum Mov Sci 10:3–53

Bullock D, Cisek P, Grossberg S (1998) Cortical networks for control of voluntary arm movements under variable force conditions. Cereb Cortex 8:48–62

Cao Y, Grossberg S (2005) A laminar cortical model of stereopsis and 3D surface perception: closure and da Vinci stereopsis. Spat Vis 18:515–578

Cao Y, Grossberg S (2012) Stereopsis and 3D surface perception by spiking neurons in laminar cortical circuits: a method of converting neural rate models into spiking models. Neural Netw 26:75–98

Cao Y, Grossberg S, Markowitz J (2011) How does the brain rapidly learn and reorganize view- and positionally-invariant object representations in inferior temporal cortex? Neural Netw 24:1050–1061

Carpenter GA (1994) A distributed outstar network for spatial pattern learning. Neural Netw 7:159–168

Carpenter GA (1997) Distributed learning, recognition, and prediction by ART and ARTMAP neural networks. Neural Netw 10:1473–1494

Carpenter GA (2001) Neural network models of learning and memory: leading questions and an emerging framework. Trends Cogn Sci 5:114–118

Carpenter GA, Gaddam SC (2010) Biased ART: a neural architecture that shifts attention toward previously disregarded features following an incorrect prediction. Neural Netw 23:435–451

Carpenter GA, Grossberg S (1987) A massively parallel architecture for a self-organizing neural pattern recognition machine. Comput Vis Graph Image Process 37:54–115

Carpenter GA, Grossberg S (1990) ART 3: hierarchical search using chemical transmitters in self-organizing pattern recognition architectures. Neural Netw 4: 129–152

Carpenter G, Grossberg S (1993) Normal and amnesic learning, recognition, and memory by a neural model of cortico-hippocampal interactions. Trends Neurosci 16:131–137

Carpenter GA, Markuzon N (1998) ARTMAP-IC and medical diagnosis: instance counting and inconsistent cases. Neural Netw 11:323–336

Carpenter GA, Grossberg S, Reynolds JH (1991a) ARTMAP: supervised real-time learning and classification of nonstationary data by a self-organizing neural network. Neural Netw 4:565–588

Carpenter GA, Grossberg S, Rosen DB (1991b) Fuzzy ART: fast stable learning and categorization of analog patterns by an adaptive resonance system. Neural Netw 4:759–771

Carpenter GA, Grossberg S, Markuzon N, Reynolds JH, Rosen DB (1992) Fuzzy ARTMAP: a neural network architecture for incremental supervised learning of analog multidimensional maps. IEEE Trans Neural Netw 3:698–713

Carpenter GA, Martens S, Ogas OJ (2005) Self-organizing information fusion and hierarchical knowledge discovery: a new framework using ARTMAP neural networks. Neural Netw 18:287–295

Chandler B, Grossberg S (2012) Joining distributed pattern processing and homeostatic plasticity in recurrent on-center off-surround shunting networks: noise, saturation, short-term memory, synaptic scaling, and BDNF. Neural Netw 25: 21–29

Chang H-C, Grossberg S, Cao Y (2014) Where's Waldo? How perceptual cognitive, and emotional brain processes cooperate during learning to categorize and find desired objects in a cluttered scene. Front Integr Neurosci doi:10.3389/fnint.2014.0043

Chapelle O, Schölkopf B, Zien A (eds) (2006) Semi-supervised learning. MIT, Cambridge

Contreras-Vidal JL, Grossberg S, Bullock D (1997) A neural model of cerebellar learning for arm movement control: cortico-spino-cerebellar dynamics. Learn Mem 3:475–502

Dranias M, Grossberg S, Bullock D (2008) Dopaminergic and non-dopaminergic value systems in conditioning and outcome-specific revaluation. Brain Res 1238:239–287

Elder D, Grossberg S, Mingolla M (2009) A neural model of visually guided steering, obstacle avoidance, and route selection. J Exp Psychol Hum Percept Perform 35:1501–1531

Fang L, Grossberg S (2009) From stereogram to surface: how the brain sees the world in depth. Spat Vis 22:45–82

Fazl A, Grossberg S, Mingolla E (2009) View-invariant object category learning, recognition, and search: how spatial and object attention are coordinated using surface-based attentional shrouds. Cogn Psychol 58:1–48

Foley NC, Grossberg S, Mingolla E (2012) Neural dynamics of object-based multifocal visual spatial attention and priming: object cueing, useful-field-of-view, and crowding. Cognitive Psychology 65: 77–117

Gancarz G, Grossberg G (1999) A neural model of the saccadic eye movement control explains task-specific adaptation. Vis Res 39:3123–3143

Grossberg S (1984) Some psychophysiological and pharmacological correlates of a developmental, cognitive, and motivational theory. In: Karrer R, Cohen J, Tueting P (eds) Brain and information: event related potential. New York Academy of Sciences, New York, pp 58–142.

Grossberg S (2000a) The complementary brain: unifying brain dynamics and modularity. Trends Cogn Sci 4:233–246

Grossberg S (2000b) The imbalanced brain: from normal behavior to schizophrenia. Biol Psychiatry 48:81–98

Grossberg S (2013) Adaptive resonance theory: how a brain learns to consciously attend, learn, and recognize a changing World. Neural Netw 37:1–47

Grossberg, S. (2016). Towards solving the hard problem of consciousness: the varieties of brain resonances and the conscious experiences that they support. Submitted for publication

Grossberg S, Huang T-R (2009) ARTSCENE: a neural system for natural scene classification. J Vis 9(6):1–19

Grossberg S, Merrill JWL (1992) A neural network model of adaptively timed reinforcement learning and hippocampal dynamics. Cogn Brain Res 1:3–38

Grossberg S, Merrill JWL (1996) The hippocampus and cerebellum in adaptively timed learning, recognition, and movement. J Cogn Neurosci 8:257–277

Grossberg S, Pearson L (2008) Laminar cortical dynamics of cognitive and motor working memory, sequence learning and performance: toward a unified theory of how the cerebral cortex works. Psychol Rev 115:677–732

Grossberg S, Pilly PK (2012) How entorhinal grid cells may learn multiple spatial scales from a dorsoventral gradient of cell response rates in a self-organizing map. PLoS Comput Biol 8(10):31002648. doi:10.1371/journal.pcbi.1002648

Grossberg S, Pilly PK (2014) Coordinated learning of grid cell and place cell spatial and temporal properties: multiple scales, attention, and oscillations. Philos Trans R Soc B 369:20120524

Grossberg S, Rudd ME (1992) Cortical dynamics of visual motion perception: short-range and long-range apparent motion (with Rudd ME). Psychol Rev 99:78–121

Grossberg S, Schmajuk NA (1989) Neural dynamics of adaptive timing and temporal discrimination during associative learning. Neural Netw 2:79–102

Grossberg S, Seidman D (2006) Neural dynamics of autistic behaviors: cognitive, emotional, and timing substrates. Psychol Rev 113:483–525

Grossberg S, Swaminathan G (2004) A laminar cortical model for 3D perception of slanted and curved surfaces and of 2D images: development, attention and bistability. Vis Res 44:1147–1187

Grossberg S, Vladusich T (2010) How do children learn to follow gaze, share joint attention, imitate their teachers, and use tools during social interactions? Neural Netw 23:940–965

Grossberg S, Leveille J, Versace M (2011) How do object reference frames and motion vector decomposition emerge in laminar cortical circuits? Atten Percept Psychophys 73:1147–1170

Grossberg S, Kuhlmann L, Mingolla E (2007) A neural model of 3D shape-from-texture: multiple-scale filtering, boundary grouping, and surface filling-in. Vis Res 47:634–672

Grossberg S, Mingolla E, Viswanathan L (2001) Neural dynamics of motion integration and segmentation within and across apertures. Vis Res 41:2521–2553

Grossberg S, Srihasam K, Bullock D (2012) Neural dynamics of saccadic and smooth pursuit eye movement coordination during visual tracking of unpredictably moving targets. Neural Netw 27:1–20

Huang T-R, Grossberg S (2010) Cortical dynamics of contextually cued attentive visual learning and search: spatial and object evidence accumulation. Psychol Rev 117:1080–1112

Hurvich LM, Jameson D (1957) An opponent-process theory of color vision. Psychol Rev 64: 384–390

Kelly FJ, Grossberg S (2000) Neural dynamics of 3-D surface perception: figure-ground separation and lightness perception. Percept Psychophys 62:1596–1619

Mhatre H, Gorchetchnikov A, Grossberg S (2012) Grid cell hexagonal patterns formed by fast self-organized learning within entorhinal cortex. Hippocampus 22:320–334

Pack C, Grossberg S, Mingolla E (2001) A neural model of smooth pursuit control and motion perception by cortical area MST. J Cogn Neurosci 13:102–120

Pilly PK, Grossberg S (2012) How do spatial learning and memory occur in the brain? Coordinated learning of entorhinal grid cells and hippocampal place cells. J Cogn Neurosci 24:1031–1054

Pilly PK, Grossberg S (2014) How does the modular organization of entorhinal grid cells develop? Front Hum Neurosci. doi:10.3389/fnhum.2014.0037

Schiller PH (1982) Central connections of the retinal ON and OFF pathways. Nature 297:580–583

Simone G, Farina A, Morabito FC, Serpico SB, Bruzzone L (2002) Image fusion techniques for remote sensing applications. Inf Fusion 3:3–15

Srihasam K, Bullock D, Grossberg S (2009) Target selection by frontal cortex during coordinated saccadic and smooth pursuit eye movements. J Cogn Neurosci 21:1611–1627

Adaptive System

▶ Complexity in Adaptive Systems

Agent

In computer science, the term "agent" usually denotes a software abstraction of a real entity which is capable of acting with a certain degree of autonomy. For example, in artificial societies, agents are software abstractions of real people, interacting in an artificial, simulated environment. Various authors have proposed different definitions of agents. Most of them would agree on the following set of agent properties:

- Persistence: Code is not executed on demand but runs continuously and decides autonomously when it should perform some activity.
- Social ability: Agents are able to interact with other agents.
- Reactivity: Agents perceive the environment and are able to react.
- Proactivity: Agents exhibit goal-directed behavior and can take the initiative.

Agent-Based Computational Models

▶ Artificial Societies

Agent-Based Modeling and Simulation

▶ Artificial Societies

Agent-Based Simulation Models

▶ Artificial Societies

AIS

▶ Artificial Immune Systems

Algorithm Evaluation

Geoffrey I. Webb
Faculty of Information Technology, Monash University, Victoria, Australia

Definition

Algorithm evaluation is the process of assessing a property or properties of an algorithm.

Motivation and Background

It is often valuable to assess the efficacy of an algorithm. In many cases, such assessment is relative, that is, evaluating which of several alternative algorithms is best suited to a specific application.

Processes and Techniques

Many machine learning and data mining algorithms have been proposed. In order to understand the relative merits of these alternatives, it is necessary to evaluate them. The primary approaches to evaluation can be characterized as either theoretical or experimental. Theoretical evaluation uses formal methods to infer properties of the algorithm, such as its computational complexity (Papadimitriou 1994), and also employs the tools of computational learning theory to assess learning theoretic properties. Experimental evaluation applies the algorithm to learning tasks to study its performance in practice.

There are many different types of property that may be relevant to assess depending upon the intended application. These include algorithmic properties, such as time and space complexity. These algorithmic properties are often assessed separately with respect to performance when learning a ▸ model, that is, at ▸ training time, and performance when applying a learned model, that is, at ▸ test time.

Other types of property that are often studied are the properties of the models that are learned (see ▸ Model Evaluation). Strictly speaking, such properties should be assessed with respect to a specific application or class of applications. However, much machine learning research includes experimental studies in which algorithms are compared using a set of data sets with little or no consideration given to what class of applications those data sets might represent. It is dangerous to draw general conclusions about relative performance in general across any application from relative performance on this sample of some unknown class of applications. Such experimental evaluation has become known disparagingly as a *bake-off*.

An approach to experimental evaluation that may be less subject to the limitations of bake-offs is the use of experimental evaluation to assess a learning algorithm's ▸ bias and variance profile. Bias and variance measure properties of an algorithm's propensities in learning models rather than directly being properties of the models that are learned. Hence, they may provide more general insights into the relative characteristics of alternative algorithms than do assessments of the performance of learned models on a finite number of applications. One example of such use of bias–variance analysis is found in Webb (2000).

Techniques for experimental algorithm evaluation include ▸ bootstrap sampling, ▸ cross-validation, ▸ holdout evaluation, ▸ out-of-sample evaluation and ▸ prospective evaluation.

Cross-References

▸ Evaluation of Learning Algorithms
▸ Model Evaluation

References

Hastie T, Tibshirani R, Friedman JH (2001) The elements of statistical learning. Springer, New York

Mitchell TM (1997) Machine learning. McGraw-Hill, New York

Papadimitriou CH (1994) Computational complexity. Addison-Wesley, Reading

Webb GI (2000) MultiBoosting: a technique for combining boosting and wagging. Mach Learn 40(2):159–196

Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, San Francisco

Analogical Reasoning

▸ Instance-Based Learning

Analysis of Text

▸ Text Mining

Analytical Learning

▸ Deductive Learning
▸ Explanation-Based Learning

Anomaly Detection

Varun Chandola[1], Arindam Banerjee[2], and
Vipin Kumar[2]
[1]State University of New York at Buffalo,
Buffalo, NY, USA
[2]University of Minnesota, Minneapolis, MN,
USA

Abstract

Anomalies correspond to the behavior of a system which does not conform to its expected or normal behavior. Identifying such anomalies from observed data, or the task of anomaly detection, is an important and often critical analysis task. This includes finding abnormalities in a medical image, fraudulent transactions in a credit card history, or structural defects in an aircraft's engine. The importance of this problem has resulted in a large body of literature on this topic. However, given that the definition of an anomaly is strongly tied to the underlying application, the existing research is often embedded in the application domains, and it is unclear how methods developed for one domain would perform in another. The goal of this article is to provide a general introduction of the anomaly detection problem. We start with the basic formulation of the problem and then discuss the various extensions. In particular, we discuss the challenges associated with identifying anomalies in structured data and provide an overview of existing research in this area. We hope that this article will provide a better understanding of the different directions in which research has been done on this topic, and how techniques developed in one area can be applied in domains for which they were not intended to begin with.

Introduction

Anomalies are the unusual, unexpected, surprising patterns in the observed world. Identifying, understanding, and predicting anomalies from data form one of the key pillars of modern data mining. Effective detection of anomalies allows extracting critical information from data which can then be used for a variety of applications, such as to stop malicious intruders, detect and repair faults in complex systems, and better understand the behavior of natural, social, and engineered systems.

Anomaly detection refers to the problem of finding anomalies in data. While "anomaly" is a generally accepted term, other synonyms, such as outliers, discordant observations, exceptions, aberrations, surprises, peculiarities, or contaminants, are often used in different application domains. In particular, anomalies and outliers are often used interchangeably. Anomaly detection finds extensive use in a wide variety of applications such as fraud detection for credit cards, insurance or healthcare, intrusion detection for cybersecurity, fault detection in safety critical systems, and military surveillance for enemy activities. The importance of anomaly detection stems from the fact that for a variety of application domains, anomalies in data often translate to significant (and often critical) actionable insights. For example, an anomalous traffic pattern in a computer network could mean that a hacked computer is sending out sensitive data to an unauthorized destination (Kumar 2005). An anomalous remotely sensed weather variable such as temperature could imply a heat wave or cold snap or even faulty remote sensing equipment. An anomalous MRI image may indicate early signs of Alzheimer's or the presence of malignant tumors (Spence et al. 2001). Anomalies in credit card transaction data could indicate credit card or identity theft (Aleskerov et al. 1997), or anomalous readings from a spacecraft sensor could signify a fault in some component of the spacecraft (Fujimaki et al. 2005).

Anomaly detection is generally considered as a core machine learning or data mining problem, in the same vein as classification and clustering. Given the practical significance of anomalies, there has been a tremendous interest in studying this problem, starting from statistical methods proposed as early as the nineteenth century (Edgeworth 1887). Over time, a variety of anomaly detection techniques have been developed in several research communities. Many of these techniques have been specifically developed for certain application domains, while others are more generic. Several books and surveys have been published in recent years that provide an overview of the vast literature on this topic (Chandola et al. 2009; Aggarwal 2013; Hodge and Austin 2004; Chandola et al. 2012; Akoglu et al. 2015).

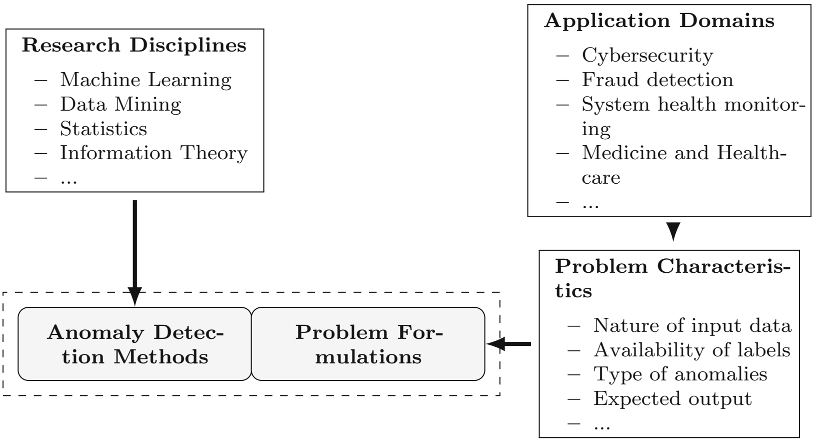
However, one key characteristic of anomaly detection sets it apart from other machine learning problems. Anomaly detection is a highly application-oriented problem which means that there is a lack of a consistent definition of an anomaly across tasks and application domains. Researchers typically define an anomaly in a way that best suits the target application. Thus, several different formulations of the anomaly detection problem exist. Existing solutions for these problem formulations have borrowed concepts from a variety of disciplines in mathematics, statistics, and computer science. This has resulted in a rich and complex landscape for anomaly detection research (See Fig. 1).

The goal of this article is to provide the readers a general understanding of the complex problem space of anomaly detection. Starting with the most basic problem setting, i.e., identifying anomalous data instances from a data set, we then discuss other formulations and the corresponding methods. To highlight the practical importance of anomaly detection, we provide an application-oriented overview of existing methods. Finally, we discuss open challenges and research questions that exist in this area to motivate future research.

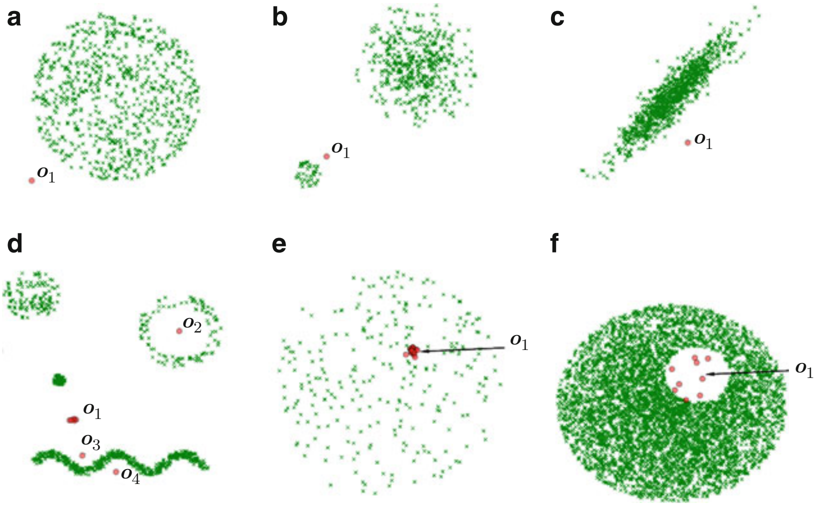
Point Anomaly Detection

In the most widely accepted setting for anomaly detection, also referred to as *point anomaly detection*, the goal is to identify points (objects, instances, etc.) in a data set that do not conform to the accepted normal behavior. Typically, no other knowledge about the normal or anomalous behavior is available. The lack of any ground truth for training makes this an *unsupervised anomaly detection problem*. In the sequel, we briefly talk about other formulations in which partial knowledge of normal and/or anomalous behavior is available.

Even in the basic setting of point anomaly detection, a uniform definition of anomaly does not exist. Figure 2 shows several hypothetical examples of anomalies in a two-dimensional data set. In each of the example, anomalies have a different interpretation. For instance, the point o_1 in Fig. 2a is anomalous because it is far away from the rest of the data points which belong to a dense region. In Fig. 2b, however, the point o_1



Anomaly Detection, Fig. 1 Anatomy of an anomaly detection problem



Anomaly Detection, Fig. 2 Examples of anomalies in 2-D data. (**a**) Anomaly with respect to rest of the data points. (**b**) Anomaly with respect to local neighborhood. (**c**) Anomaly with respect to the data distribution. (**d**) Anomaly with respect to local dense regions. (**e**) Anomalous tight cluster in a sparse region. (**f**) Anomalous sparse cluster in a dense region

is anomalous because it lies relatively far away from a dense region, even though there are points in the second sparse region which are equally distant from their nearest points. In the third example (see Fig. 2c), the point o_1 is anomalous because it lies away from the statistical distribution (bivariate normal) of the data. On the other hand, there are several points located at the ends of the elliptical distribution that are farther away from the points that o_1. The anomalies o_2, o_3, and o_4 in Fig. 2d are points that are away from their closest dense regions. The points in the anomalous set o_1 in Fig. 2d, e, and f are all groups of points whose density is anomalous with respect to the rest of the data set. As shown in the above simple 2-D example, even for point anomaly detection, one can define anomalies in multiple ways. Most existing anomaly detection methods, on the other hand, have been developed, by starting from a different notion of anomaly often motivated by a specific application domain. Thus, one method might be successful in one scenario and not in the other. We now discuss some prominent classes of point anomaly detection methods and the definitions of anomalies that they are best suited for. The various classes of point anomaly detection methods are briefly discussed below:

Nearest neighbor-based methods analyze the nearest neighborhood of a test instance to assign it an anomaly score (Ramaswamy et al. 2000; Knorr and Ng 1999; Knorr et al. 2000; Otey et al. 2006; Tang et al. 2002; Breunig et al. 2000, 1999). The key assumption underlying nearest neighbor-based anomaly detection methods is that normal points lie in dense neighborhoods and anomalous points lie in sparse neighborhoods. Nearest neighbor methods consider suitable measures of density, e.g., distance to the k-th nearest neighbor (Ramaswamy et al. 2000), radius needed to enclose a certain number of points (Knorr and Ng 1999; Knorr et al. 2000), etc. Such methods are capable of identifying global anomalies (See Fig. 2a) but are shown to perform poorly when the data has regions with varying densities (See Fig. 2b). For such scenarios, methods such as local outlier factor (Breunig et al. 1999) and commute distance-based outlier factor (Khoa and Chawla 2010) have been proposed. When data is high dimensional, such methods typically suffer from the "curse of dimensionality." Methods such as angle-based outlier detection (Kriegel et al. 2008) and subspace-based approaches (Zhang and Wang 2006) have been proposed to address this issue.

Clustering-based methods learn clusters from a given data set and assign an anomaly score to a test instance based on its relationship with its nearest cluster (Eskin et al. 2002; He et al. 2003; Marchette 1999; Eskin et al. 2002; Portnoy et al. 2001; Mahoney et al. 2003). Clustering-based methods assume that while normal points exhibit cluster structure, anomalous points do not belong to a cluster or are far away from the nearest normal cluster representative. In certain settings, if the anomalies themselves may form a cluster, one assumes that normal points form large and dense clusters, whereas anomalous points form small clusters or clusters with low density (see Fig. 2d, e and f). While such methods identify anomalies as a post-clustering phase, recently, there have been methods that focus on identifying anomalies simultaneously with the clusters (Ott et al. 2014; Chawla and Gionis 2013).

Statistical methods estimate a parametric or nonparametric model from the data and apply a statistical test on the probability of the instance to be generated by the estimated model to assign an anomaly score to the test instance (Barnett and Lewis 1994; Fox 1972; Abraham and Chuang 1989; Laurikkala et al. 2000; Chow and Yeung 2002). Such statistical models assume that normal points appear in the high probability regions of the distribution, thereby having high likelihood of occurring and hence low anomaly scores. On the other hand, anomalous points appear in the low probability regions of the distribution and have high anomaly score. Such methods are effective if the normal instances can be modeled by a statistical distribution. For instance, if the data in Fig. 2c is modeled as a bivariate normal distribution, the anomalous point o_1 can be easily identified using a standard Mahalanobis statistic, while rest of the points will appear normal.

Classification-based methods learn a classifier from a labeled (or unlabeled) training data and assign an anomaly score or label to a test data instance (Tax 2001; Tax and Duin 1999a, b; Barbara et al. 2001; Roth 2004; Hawkins et al. 2002; Mahoney and Chan 2002, 2003). The key assumption underlying classification-based anomaly detection methods is that based on the available training data, one can learn a classifier in the given feature space to distinguish between normal and anomalous points. Classification-based anomaly detection methods can be categorized into one-class methods, which have one model for the normal class and any point which does not fit that model is deemed anomalous, and multi-class methods, which have multiple normal classes and points which do not fit any of the normal classes are deemed anomalous. A variety of models such as support vector machines, neural networks, Bayesian models, and rule-based systems have been used for classification-based anomaly detection. However, such methods are limited by their dependence on availability of labels for normal and/or anomalous behavior. There are, however, methods that can operate in a purely unsupervised setting, such as the one-class support vector machines (Schölkopf et al. 2001; Tax 2001).

Spectral decomposition-based methods find an approximation of the data using a combination of attributes that capture the bulk of variability in the data. Instances that are significantly different from others in the lower approximation are detected as anomalies (Agovic et al. 2007; Parra et al. 1996; Shyu et al. 2003; Fujimaki et al. 2005). Such methods are particularly effective in scenarios where the data is being generated from a lower dimensional manifold, e.g., See Fig. 2c.

Information theoretic methods are based on the assumption that anomalies in data induce irregularities in the information content of the data set. Such methods analyze the *information content* of a data set using different information theoretic measures such as *Kolmogorov complexity, entropy, relative entropy*, etc. and detect instance that induces irregularities in the information content of the data set as anomalies (Arning et al. 1996; Keogh et al. 2004; Lee and Xiang 2001; He et al. 2005, 2006).

Extensions to Point Anomaly Detection

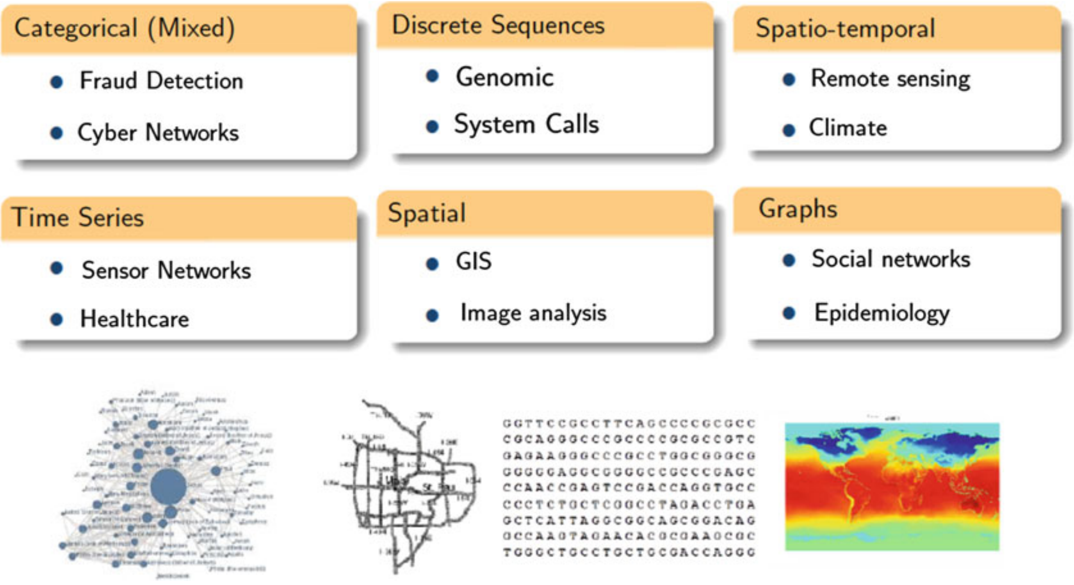
In certain settings, the unsupervised point anomaly detection problem discussed in section "Point Anomaly Detection" is not rich enough to capture all requirements of an application domain. Here we discuss some of the different ways in which the basic problem setting is typically extended.

Nature of Input Data

The modality of the data determines the applicability of anomaly detection techniques. For example, for statistical techniques, different statistical models have to be used for continuous and categorical data. Similarly, for nearest neighbor-based techniques, the nature of attributes would determine the distance measure to be used. Often, instead of the actual data, the pairwise distance between instances might be provided in the form of a distance (or similarity) matrix. In such cases, techniques that require original data instances are not applicable, e.g., many statistical methods and certain classification-based techniques. However, many of the nearest neighbor-based or clustering-based methods discussed in section "Point Anomaly Detection" are still applicable.

Input data can also be categorized based on the relationship present among data instances (Tan et al. 2005). Most of the existing anomaly detection techniques deal with data represented as a vector of attributes (record or point data, if the data can be mapped onto a coordinate space), as discussed in section "Point Anomaly Detection." Typically, no relationship is assumed among the data instances.

In general, data instances can be related to each other. Some examples are *sequence data*, *spatial data*, and *graph data* (See Fig. 3 for an overview). In sequence data, the data instances are linearly ordered, e.g., time-series data, genome sequences, protein sequences. In *spatial data*, each data instance is related to its neighboring instances, e.g., vehicular traffic data, ecological data. When the spatial data has a temporal (sequential) component, it is referred to as *spatiotemporal* data, e.g., climate data. In *graph data*, data instances are represented as vertices in a graph and are connected to other vertices with edges. Later in this section, we will



Anomaly Detection, Fig. 3 Complex data types encountered by anomaly detection and some sample application domains discuss situations where such relationship among data instances becomes relevant for anomaly detection.

Type of Anomaly

Anomaly detection techniques vary depending on the nature of the desired anomaly. We have already discussed point anomalies in section "Point Anomaly Detection," which is the most common form of anomaly. While point anomalies are isolated by nature, several applications need to consider anomalies in a context or small collection of observations which appear anomalous. One can define two additional types of anomalies to capture such structures: contextual anomalies and collective anomalies.

Contextual Anomalies

Data instances which are anomalous in a specific context, but not otherwise, are called contextual anomaly (also referred to as *conditional anomaly* Song et al. 2007). For example, a temperature of 70 °F may be normal over summer, but is anomalous in the context of winter; a heart rate of 130 may be normal for an individual exercising or running, but is anomalous when the individual is resting. In the setting of contextual anomaly detection, the context, such as summer/winter and exercising/resting, has to be specified as a part of the problem formulation. In particular, the data instances are defined using following two sets of attributes:

1. *Contextual attributes*. The contextual attributes are used to determine the context (or neighborhood) for that instance. For example, in spatial data sets, the longitude and latitude of a location are the contextual attributes. In time-series data, time is a contextual attribute which determines the position of an instance on the entire sequence.
2. *Behavioral attributes*. The behavioral attributes define the non-contextual characteristics of an instance. For example, in a spatial data set describing the average rainfall of the entire world, the amount of rainfall at any location is a behavioral attribute.

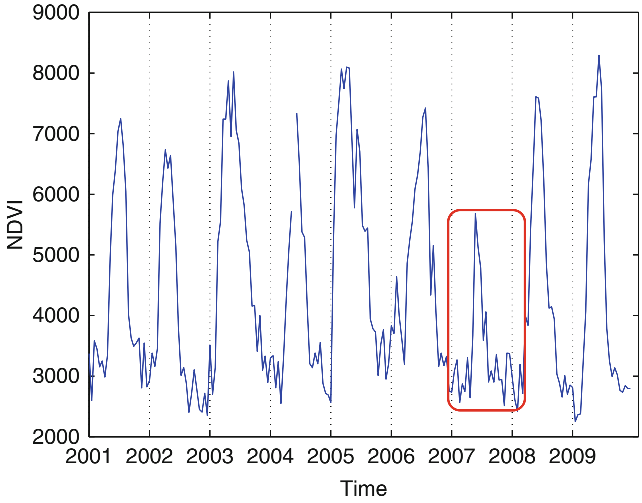
The context determines the normal behavioral attributes, and the normal can be different in different contexts. Anomalous behavior is determined using the values for the behavioral attributes within a specific context, in particular when such values deviate from what is normal in that context. A data instance might be a contextual anomaly in a given context, but an identical data instance (in terms of behavioral attributes) could be considered normal in a different context. This property is key in identifying contextual and behavioral attributes for a contextual anomaly detection technique.

Contextual anomalies have been most commonly explored in time-series data (Weigend et al. 1995; Salvador and Chan 2003) and spatial data (Kou et al. 2006; Shekhar et al. 2001). In spatial data domain, an observation has a neighborhood specified by its location component (refer to our earlier discussion on spatial data). Consider an example in which each data instance is a county location which is defined over several attributes. If these attributes show high pollution levels for a particular county, but the neighborhood of this county is also highly polluted, then this county is not an anomaly. But if the neighborhood has very low pollution, then this county becomes an anomaly.

A similar example can be found in the credit card fraud detection domain. A contextual attribute in credit card domain can be the *time* of purchase. Suppose an individual usually has a weekly shopping bill of $100 except during the Christmas week, when it reaches $1000. A new purchase of $1000 in a week in July will be considered a contextual anomaly, since it does not conform to the normal behavior of the individual in the context of time (even though the same amount spent during Christmas week will be considered normal).

The choice of applying a contextual anomaly detection technique is determined by the meaningfulness of the contextual anomalies in the target application domain. Another key factor is the availability of *contextual* attributes. In several cases, defining a context is straightforward, and hence applying a contextual anomaly detection technique makes sense. In other cases, defining

Anomaly Detection, Fig. 4 Example of a collective anomaly—MODIS NDVI Time Series for 2001–2009 for a Southern California location with a known forest fire (*Canyon fire*) in 2007 [*src:* http://cdfdata.fire.ca.gov/incidents/incidents_archived?archive_year=2007]



a context is not easy, making it difficult to apply such techniques.

Collective Anomalies

If a collection of related data instances is anomalous with respect to the entire data set, it is termed as a collective anomaly. The individual data instances in a collective anomaly may not be anomalies by themselves, but their occurrence together as a collection is anomalous. Figure 4 illustrates an example which shows a greenness measurement called normalized difference vegetation index (NDVI) for a geographic location obtained from a satellite instrument (MODIS). The highlighted region denotes an anomaly where the greenness values are abnormally low for the entire year of 2007 due to a wildfire during that time. Note that the individual measurements during the year are not anomalous by themselves.

As an another illustrative example, consider a sequence of actions occurring in a computer as shown below:

… http-web, buffer-overflow, http-web, http-web, smtp-mail, ftp, http-web, ssh, smtp-mail, http-web, **ssh**, **buffer-overflow**, **ftp**, http-web, ftp, smtp-mail,http-web …

The highlighted sequence of events (**buffer-overflow**, **ssh**, **ftp**) corresponds to a typical web-based attack by a remote machine followed by copying of data from the host computer to remote destination via *ftp*. It should be noted that this collection of events is an anomaly, but the individual events are not anomalies when they occur in other locations in the sequence.

Collective anomalies have been explored for sequence (discrete and time series) data (Forrest et al. 1999; Sun et al. 2006), graph data (Noble and Cook 2003; Li et al. 2014; Akoglu et al. 2015), and spatial data (Shekhar et al. 2001). It should be noted that while point anomalies can occur in any data set, collective anomalies can occur only in data sets in which data instances are related. In contrast, occurrence of contextual anomalies depends on the availability of context attributes in the data. A point anomaly or a collective anomaly can also be a contextual anomaly if analyzed with respect to a context. Thus a point anomaly detection problem or collective anomaly detection problem can be transformed to a contextual anomaly detection problem by incorporating the context information.

Data Labels

In some scenarios, labels associated with data instances denote if that instance is *normal* or *anomalous*. (Also referred to as normal and anomalous classes.) Obtaining labeled data which is accurate as well as representative of all types of normal and anomalous behaviors is often prohibitively expensive. Labels are often provided by human domain experts and hence usually require substantial effort and time. Even in settings where a human expert is able to provide labels, it is usually easier to give examples of normal instances, since the number of different ways an anomaly can occur is quite large and finding examples of the different types of anomalies is difficult. Further, anomalous behavior is often dynamic in nature, e.g., new types of anomalies might arise, for which there is no labeled training data. In certain cases, such as aviation safety, anomalous instances would translate to catastrophic events, and hence will be very rare.

Based on the extent to which the labels are available, anomaly detection techniques can operate in one of the following three modes: supervised, semi-supervised, and unsupervised anomaly detection. Several of the anomaly detection methods discussed in section "Point Anomaly Detection" are unsupervised methods. Semi-supervised methods typically assume availability of a training data that represents the normal behavior. The general approach for such methods is to construct a statistical or machine learning model of normal behavior and then apply a statistical or proximity test to detect new instances which are not consistent with the learnt model. Supervised methods assume availability training data that represents both normal and anomalous behavior. Typically, anomalous events have a much smaller prior probability, and one can leverage methods for rare-class classification, cost-sensitive classification, and other ways of handling class imbalance. However, such methods find limited applicability since obtaining representative training data for anomalous behavior is typically infeasible.

Output of Algorithm

An important aspect for any anomaly detection technique is the manner in which the anomalies are reported. Typically, the outputs produced by anomaly detection techniques are one of the following two types: scores or binary predictions. Scores allow analysts to rank the anomalies in terms of the severity. Typically a threshold is then applied to the scores to identify the anomalies on which to act upon. The threshold is often set by either identifying a natural cutoff point in the sorted scores or based on the number of desired anomalies for further analysis. Methods that assign a binary label to data objects (anomaly or normal) are often easier to understand, though they lack the capability of ranking the anomalies. There are some research in calibrating the scores as probabilities (Gao and Tan 2006) for better interpretability (Kriegel et al. 2011; Schubert et al. 2012).

Anomaly Detection for Complex Data

In section "Point Anomaly Detection" we discussed anomaly detection in the context of data without any explicit relationship defined among them. However, in many applications, data objects are related, and often the anomalous behavior can only be identified by analyzing the relationships between the objects. In this section, we discuss the anomaly detection methods developed to handle the different types of relationships.

Symbolic Sequences

In this section, we provide an overview of the existing research on anomaly detection for symbolic sequences. Methods in this area can be grouped into following categories:

- **Kernel-Based Techniques:** These techniques treat the entire test sequence as a unit element in the analysis (Budalakoti et al. 2006, 2007; Yang and Wang 2003) and hence are analogous to point-based anomaly detection techniques. They typically apply a proximity-based point anomaly detection technique by defining an appropriate similarity kernel for the sequences.
- **Window-Based Techniques:** These techniques analyze a short window of symbols—a short subsequence—within the test sequence at a time (Forrest et al. 1996; Hofmeyr et al. 1998; Endler 1998; Debar et al. 1998; Ghosh et al. 1999a, b; Lane and Brodley 1997, 1999; Cabrera et al. 2001). Thus such techniques treat a subsequence within the test sequence as a unit element for analysis. These techniques require an additional step

in which the anomalous nature of the entire test sequence is determined, based on the analysis on the subsequences within the entire sequence.

- **Markovian Techniques:** These techniques predict the probability of observing each symbol of the test sequence, using a probabilistic model, and use the per-symbol probabilities to obtain an anomaly score for the test sequence (Sun et al. 2006; Ye 2004; Michael and Ghosh 2000; Eskin et al. 2001; Lee et al. 1997). These techniques analyze each symbol with respect to previous few symbols.
- **Hidden Markov Model-Based Techniques:** These techniques transform the input sequences into sequences of hidden states and then detect anomalies in the transformed sequences (Forrest et al. 1999; Qiao et al. 2002; Zhang et al. 2003; Florez-Larrahondo et al. 2005).

Though several techniques have been proposed for symbolic sequences in various application domains, there has not been any cross domain evaluation and understanding of the existing techniques. Forrest et al. (1999) compared four different anomaly detection techniques, but evaluated them in the context of system call intrusion detection. Sun et al. (2006) proposed a technique for protein sequences, but no evaluation with techniques proposed for system call data was done. Similarly, while Budalakoti et al. (2006) proposed a clustering-based techniques to detect anomalies in flight sequences, it has not been shown how the same technique would perform on system call intrusion detection data or protein data.

Most of the above methods identify an anomalous sequence from a set of sequences, assuming that majority of the sequences are normal. Other methods focus on a different problem formulation, also referred to as *discord detection*, in which the goal is to identify a subsequence within a long sequence which is anomalous with respect to the rest of the sequence. Most of the existing techniques that handle this problem formulation slide a fixed length window across the given long sequence and compare each window with the remaining sequence to detect anomalous windows (Keogh et al. 2005a, 2006; Lin et al. 2005; Wei et al. 2005).

Time Series

Most methods that handle time-series data deal primarily with univariate signals, i.e., a single measurement captured over time. Several statistical techniques detect anomalous observations (also referred to as *outliers*) within a single time series using various time series modeling techniques such as regression (Fox 1972; Abraham and Chuang 1989; Rousseeuw and Leroy 1987), autoregression (AR) (Fujimaki et al. 2005; Wu and Shao 2005), ARMA (Pincombe 2005), ARIMA (Zare Moayedi and Masnadi-Shirazi 2008), support vector regression (SVR) (Ma and Perkins 2003), Kalman filters Knorn and Leith (2008), etc. The general approach behind such techniques is to forecast the next observation in the time series, using the statistical model and the time series observed so far, and compare the forecasted observation with the actual observation to determine if an anomaly has occurred.

Two broad categories of techniques have been proposed to identify anomalous time series in a time-series database (Chandola et al. 2009), viz., segmentation-based and kernel-based anomaly detection techniques. The general approach behind segmentation-based techniques is to segment the normal time series and treat each segment as a state in a *finite-state automaton* (FSA) and then use the FSA to determine if a test time series is anomalous or not. Several variants of the segmentation-based technique have been proposed (Chan and Mahoney 2005; Mahoney and Chan 2005; Salvador and Chan 2005). Kernel-based anomaly detection techniques compute similarity/distance between time series and apply a nearest neighbor-based anomaly detection technique on the similarity "kernel" (Protopapas et al. 2006; Wei et al. 2006; Yankov et al. 2007). Protopapas et al. (2006) use *cross correlation* as the similarity measure and compute the anomaly score of a test time series as the inverse of its average similarity to all other time series in the given data set. Wei et al. (2006) use a *rotation invariant* version of Euclidean distance to compute distance between time series and then assign an anomaly score to each time series as equal to its distance to its nearest neighbor. Yankov et al. (2007) proposed pruning-based heuristics to improve the efficiency of the nearest neighbor technique (Wei et al. 2006).

Several anomaly detection techniques for time series data identify anomalous subsequences within a long time series (also referred to as *discords*) (Keogh et al. 2004, 2005a, 2006; Lin et al. 2005; Fu et al. 2006; Bu et al. 2007; Yankov et al. 2007). Such techniques analyze fixed length windows obtained from the time series by comparing each window with the rest of the time series or against all other windows from that time series. A window which is significantly different from other windows is declared as a discord.

Limited research has been done to identify anomalies in multivariate time series data. Most existing methods for multivariate time series focus on detecting a single anomalous multivariate observation (Baragona and Battaglia 2007; Galeano et al. 2004; Tsay et al. 2000). Baragona and Battaglia (2007) propose an ICA-based technique to detect outliers in multivariate time series. The underlying idea is to isolate the multivariate time series into a set of independent univariate components and an outlier signal and analyze the univariate outlier signal to determine the outliers. The ICA-based technique assumes that the observed signals are linear combination of independent components as well as independent noise signal, and the added noise has a high kurtosis.

Cheng et al. (2009) proposed a distance-based approach to detect anomalous subsequences within a given multivariate sequence. For a given multivariate sequence S, all w length windows are extracted. The distance between each pair of windows is computed to obtain a symmetric $(T - w + 1) \times (T - w + 1)$ kernel matrix. A fully connected graph is constructed using the kernel matrix in which each node represents a w length window and the weight on the edges between the pair of windows is equal to the similarity (inverse of distance) between the pair. The nodes (or components) of the graph that have least connectivity are declared as anomalies.

Graphs and Networks

There has been considerable work done in the area of anomaly detection in graphs (Akoglu et al. 2015). Two broad categories of methods exist for detecting anomalies in graphs. The first type of methods looks for anomalous substructures or patterns within a graph (collective anomalies), while the second type of methods focuses on identifying anomalous nodes (contextual anomalies).

The first type of methods typically operates on graphs in which the nodes and/or edges are described using a set of attributes. Such graphs are often referred to as attributed graphs. The general approach here is to identify subgraphs within a large graph that have similar distribution of attributes (Noble and Cook 2003; Eberle and Holder 2007). In particular, the work by Noble and Cook (2003) identifies the frequent subgraphs in a graph with categorical attributes. Any subgraph that does not match the frequent subgraphs is considered to be anomaly. Subsequently, several variants of the original method have been proposed (Gao et al. 2010; Li et al. 2014; Sánchez et al. 2014).

The second type of methods analyzes each node in a graph with respect to its neighborhood. For instance, the OddBall method (Akoglu et al. 2010), analyzes each node with respect to its *egonet* which is the subgraph induced by the node and the other nodes connected to it in the graph. Other similar methods identify nodes that do not belong to densely connected communities (Sun et al. 2005; Ding et al. 2012; Tong and Lin 2011). Similar methods have been proposed to identify anomalies in attributed graphs (Gao et al. 2010; Müller et al. 2013).

Conclusions and Future Directions

The notion of anomaly is important in most real world settings. Data-driven methods for timely and effective identification of anomalies are essential, and this has triggered tremendous interest in the research community. However, the key difference between anomaly detection and other machine learning problems such as classification and clustering is the lack of a consistent definition of anomalies. Instead, several definitions of anomalies exist, each tailored to the need of an underlying application. In this article, we have provided an overview of this rich area by discussing the key aspects of this problem.

We have discussed different classifications of anomaly detection methods and provided understanding of the strengths and weaknesses of these classes of methods. One of the important subareas in this context is the class of methods that handle complex structured data. We have discussed methods that have been specifically developed to handle sequences, time series, and network data.

Given the unstructured nature of current research, a theoretical understanding of the anomaly detection is challenging to obtain. A possible future work would be to unify the assumptions made by different techniques regarding the normal and anomalous behavior into a statistical or machine learning framework. A limited attempt in this direction is provided by Knorr et al. (1997), where the authors show the relation between distance based and statistical anomalies for two-dimensional data sets.

There are several promising directions for further research in anomaly detection. Contextual and collective anomaly detection techniques are beginning to find increasing applicability in several domains, and there is much scope for development of new techniques in this area. The presence of data across different distributed locations has motivated the need for distributed anomaly detection techniques (Zimmermann and Mohay 2006). While such techniques process information available at multiple sites, they often have to simultaneously protect the information present in each site, thereby requiring privacy preserving anomaly detection techniques (Vaidya and Clifton 2004). With the emergence of sensor networks, processing data as it arrives has become a necessity. Many techniques discussed in this article require the entire test data before detecting anomalies. Recently, techniques have been proposed that can operate in an online fashion (Pokrajac et al. 2007); such techniques not only assign an anomaly score to a test instance as it arrives, but also incrementally update the model.

References

Abraham B, Chuang A (1989) Outlier detection and time series modeling. Technometrics 31(2):241

Aggarwal CC (2013) Outlier analysis, Springer, New York

Agovic A, Banerjee A, Ganguly AR, Protopopescu V (2007) Anomaly detection in transportation corridors using manifold embedding. In: First international workshop on knowledge discovery from sensor data, ACM Press, New York

Akoglu L, McGlohon M, Faloutsos C (2010) OddBall: spotting anomalies in weighted graphs. In: In Pacific-Asia conference on knowledge discovery and data mining (PAKDD), Hyderabad

Akoglu L, Tong H, Koutra D (2015) Graph based anomaly detection and description: a survey. Data Min Knowl Discov 29(3):626

Aleskerov E, Freisleben B, Rao B (1997) Cardwatch: a neural network based database mining system for credit card fraud detection. In: Proceedings of IEEE computational intelligence for financial engineering, New York, pp 220–226

Arning A, Agrawal R, Raghavan P (1996) A linear method for deviation detection in large databases. In: Proceedings of 2nd international conference of knowledge discovery and data mining, pp 164–169. citeseer.ist.psu.edu/arning96linear.html

Baragona R, Battaglia F (2007) Outliers detection in multivariate time series by independent component analysis. Neural Comput 19(7):1962. doi:http://dx.doi.org/10.1162/neco.2007.19.7.1962

Barbara D, Couto J, Jajodia S, Wu N (2001) Detecting novel network intrusions using bayes estimators. In: Proceedings of the first SIAM international conference on data mining, Chicago

Barnett V, Lewis T (1994) Outliers in statistical data, Wiley, Chichester

Breunig MM, Kriegel HP, Ng RT, Sander J (1999) Optics-of: identifying local outliers. In: Proceedings of the third European conference on principles of data mining and knowledge discovery, Springer, Berlin/New York, pp 262–270

Breunig MM, Kriegel HP, Ng RT, Sander J (2000) LOF: identifying density-based local outliers. In: Proceedings of 2000 ACM SIGMOD international conference on management of data. ACM Press, pp 93–104. doi:http://doi.acm.org/10.1145/342009.335388

Bu Y, Leung TW, Fu A, Keogh E, Pei J, Meshkin S (2007) WAT: finding top-k discords in time series database. In: Proceedings of 7th siam international conference on data mining

Budalakoti S, Srivastava A, Akella R, Turkov E (2006) Anomaly detection in large sets of high-dimensional symbol sequences. Technical report NASA TM-2006-214553, NASA Ames Research Center

Budalakoti S, Srivastava A, Otey M (2007) Anomaly detection and diagnosis algorithms for discrete symbol sequences with applications to airline safety. In: Proceedings of the IEEE international conference on systems, man, and cybernetics, Montreal, vol. 37

Cabrera JBD, Lewis L, Mehra RK (2001) Detection and classification of intrusions and faults using sequences of system calls. SIGMOD Records 30(4):25. doi:http://doi.acm.org/10.1145/604264.604269

Chan PK, Mahoney MV (2005) Modeling multiple time series for anomaly detection. In: Proceedings of the fifth IEEE international conference on data mining. IEEE Computer Society, Washington, DC, pp 90–97

Chandola V, Banerjee A, Kumar V (2009) Anomaly detection a survey. ACM Comput Surv 41(3):15:1–15:58

Chandola V, Banerjee A, Kumar V (2012) Anomaly detection for discrete sequences: a survey. IEEE Trans Knowl Data Eng 24:823. doi:http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.235

Chandola V, Cheboli D, Kumar V (2009) Detecting anomalies in a timeseries database. Technical report 09-004, Computer Science Department, University of Minnesota

Chawla S, Gionis A (2013) k-means-: a unified approach to clustering and outlier detection. In: Proceedings of the 13th SIAM international conference on data mining, Austin, 2–4 May 2013, pp 189–197

Cheng H, Tan PN, Potter C, Klooster S (2009) Detection and characterization of anomalies in multivariate time series. In: Proceedings of the ninth SIAM international conference on data mining (SDM)

Chow C, Yeung DY (2002) Parzen-window network intrusion detectors. In: Proceedings of the 16th International conference on pattern recognition, vol 4. IEEE Computer Society, Washington, DC, p 40385

Debar H, Dacier M, Nassehi M, Wespi A (1998) Fixed vs. variable-length patterns for detecting suspicious process behavior. In: Proceedings of the 5th European symposium on research in computer security, Springer, London, pp 1–15

Ding Q, Katenka N, Barford P, Kolaczyk E, Crovella M (2012) Intrusion as (anti)social communication: characterization and detection. In: Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining (KDD'12), pp 886–894

Eberle W, Holder L (2007) Anomaly detection in data represented as graphs. Intell Data Anal 11(6):663. http://dl.acm.org/citation.cfm?id=1368018.1368024

Edgeworth FY (1887) On discordant observations. Philos Mag 23(5):364

Endler D (1998) Intrusion detection: applying machine learning to solaris audit data. In: Proceedings of the 14th annual computer security applications conference. IEEE Computer Society, Los Alamitos, p 268

Eskin E, Arnold A, Prerau M, Portnoy L, Stolfo S (2002) A geometric framework for unsupervised anomaly detection. In: Proceedings of applications of data mining in computer security. Kluwer Academics, Dordrecht, pp 78–100

Eskin E, Lee W, Stolfo S (2001) Modeling system call for intrusion detection using dynamic window sizes. In: Proceedings of DISCEX. citeseer.ist.psu.edu/portnoy01intrusion.html

Florez-Larrahondo G, Bridges SM, Vaughn R (2005) Efficient modeling of discrete events for anomaly detection using hidden Markov models. Inf Secur 3650:506

Forrest S, Hofmeyr SA, Somayaji A, Longstaff TA (1996) A sense of self for unix processes. In: Proceedings of the ISRSP'96, pp 120–128. citeseer.ist.psu.edu/forrest96sense.html

Forrest S, Warrender C, Pearlmutter B (1999) Detecting intrusions using system calls: alternate data models. In: Proceedings of the 1999 IEEE ISRSP. IEEE Computer Society, Washington, DC, pp 133–145

Fox AJ (1972) Outliers in time series. J R Stat Soc Ser. B(Methodolog) 34(3):350

Fu AWC, Leung OTW, Keogh EJ, Lin J (2006) Finding time series discords based on haar transform. In: Proceeding of the 2nd International conference on advanced data mining and applications. Springer, Berlin/New York, pp 31–41

Fujimaki R, Yairi T, Machida K (2005) An anomaly detection method for spacecraft using relevance vector learning. In: Proceeding of the eleventh ACM SIGKDD international conference on knowledge discovery in data mining. ACM Press, New York, pp 401–410. doi:http://doi.acm.org/10.1145/1081870.1081917

Fujimaki R, Yairi T, Machida K (2005) An approach to spacecraft anomaly detection problem using kernel feature space. Adv Knowl Discov Data Min 3518:785

Galeano P, Pena D, Tsay RS (2004) Outlier detection in multivariate time series via projection pursuit. Statistics and Econometrics Working Papers ws044211, Universidad Carlos III, Departamento de Estadïstica y Econometrïca

Gao J, Tan PN (2006) Converting output scores from outlier detection algorithms into probability estimates. In: Proceedings of the sixth international conference on data mining (ICDM '06), Hong Kong, pp 212–221

Gao J, Liang F, Fan W, Wang C, Sun Y, Han J (2010) On community outliers and their efficient detection in information networks. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining (KDD '10), Washington, DC, pp 813–822

Ghosh AK, Schwartzbard A, Schatz M (1999) Learning program behavior profiles for intrusion detection. In: Proceedings of SANS third conference and workshop on intrusion detection and response. citeseer.ist.psu.edu/ghosh99learning.html

Ghosh AK, Schwartzbard A, Schatz M (1999) Using program behavior profiles for intrusion detection. In: Proceedings of 1st USENIX workshop on intrusion detection and network monitoring, Santa Clara, pp 51–62

Hawkins S, He H, Williams GJ, Baxter RA (2002) Outlier detection using replicator neural networks. In: Proceedings of the 4th international conference on data warehousing and knowledge discovery. Springer, Berlin, pp 170–180

He Z, Deng S, Xu X, Huang JZ (2006) A fast greedy algorithm for outlier mining. In: Proceedings of 10th Pacific-Asia conference on knowledge and data discovery, pp 567–576

He Z, Xu X, Deng S (2003) Discovering cluster-based local outliers. Pattern Recognit Lett 24(9–10):1641. doi:http://dx.doi.org/10.1016/S0167-8655(03)00003-5

He Z, Xu X, Deng S (2005) An optimization model for outlier detection in categorical data. In: Proceedings of international conference on intelligent computing, vol 3644. Springer, Berlin/Heidelberg

Hodge V, Austin J (2004) A survey of outlier detection methodologies. Artif Intell Rev 22(2):85. doi:http://dx.doi.org/10.1023/B:AIRE.0000045502.10941.a9

Hofmeyr SA, Forrest S, Somayaji A (1998) Intrusion detection using sequences of system calls. J Comput Secur 6(3):151. citeseer.ist.psu.edu/hofmeyr98intrusion.html

Keogh E, Lin J, Fu A (2005) Hot sax: Efficiently finding the most unusual time series subsequence. In: Proceedings of the fifth IEEE international conference on data mining, IEEE Computer Society, Washington, DC, pp 226–233. doi:http://dx.doi.org/10.1109/ICDM.2005.79

Keogh E, Lin J, Lee SH, Herle HV (2006) Finding the most unusual time series subsequence: algorithms and applications. Knowl Inf Syst 11(1):1. doi:http://dx.doi.org/10.1007/s10115-006-0034-6

Keogh E, Lonardi S, Ratanamahatana CA (2004) Towards parameter-free data mining. In: Proceedings of the 10th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM Press, New York, pp 206–215. doi:http://doi.acm.org/10.1145/1014052.1014077

Khoa NLD, Chawla S (2010) Robust outlier detection using commute time and eigenspace embedding. In: Advances in knowledge discovery and data mining, 14th Pacific-Asia conference, PAKDD 2010. Proceedings, Part II. Hyderabad, 21–24 June 2010, pp 422–434

Knorn F, Leith D (2008) Adaptive Kalman filtering for anomaly detection in software appliances. In: IEEE INFOCOM workshops 2008, Phoenix, AZ, pp 1–6

Knorr EM, Ng RT (1997) A unified approach for mining outliers. In: Proceedings of the 1997 conference of the centre for advanced studies on collaborative research. IBM Press, Toronto, p 11

Knorr EM, Ng RT (1999) Finding intensional knowledge of distance-based outliers. In: The VLDB journal, pp 211–222. citeseer.ist.psu.edu/knorr99finding.html

Knorr EM, Ng RT, Tucakov V (2000) Distance-based outliers: algorithms and applications. VLDB J 8(3–4):237. doi:http://dx.doi.org/10.1007/s007780050006

Kou Y, Lu CT, Chen D (2006) Spatial weighted outlier detection. In: Proceedings of SIAM conference on data mining, Bethesda

Kriegel HP, Hubert MS, Zimek A (2008) Angle-based outlier detection in highdimensional data. In: Proceedings of the 14th ACM SIGKDD international conference on knowledge discovery and data mining (KDD '08), Las Legas, pp 444–452

Kriegel HP, Krger P, Schubert E, Zimek A (2011) Interpreting and unifying outlier scores. In: SDM. SIAM/Omnipress, Mesa, AZ, USA, pp 13–24

Kumar V (2005) Parallel and distributed computing for cybersecurity. Distributed systems online. IEEE 6(10). doi:10.1109/MDSO.2005.53

Lane T, Brodley CE (1997) Sequence matching and learning in anomaly detection for computer security. In: Fawcett T, Haimowitz I, Provost F, Stolfo S (eds) Proceedings of AI approaches to fraud detection and risk management. AAAI Press, Menlo Park, pp 43–49

Lane T, Brodley CE (1999) Temporal sequence learning and data reduction for anomaly detection. ACM Trans Inf Syst Secur 2(3):295. doi:http://doi.acm.org/10.1145/322510.322526

Laurikkala J, Juhola1 M, Kentala E (2000) Informal identification of outliers in medical data. In: Fifth international workshop on intelligent data analysis in medicine and pharmacology, Berlin, pp 20–24

Lee W, Xiang D (2001) Information-theoretic measures for anomaly detection. In: Proceedings of the IEEE symposium on security and privacy. IEEE Computer Society, Washington, DC, p 130

Lee W, Stolfo S, Chan P (1997) Learning patterns from unix process execution traces for intrusion detection. In: Proceedings of the AAAI 97 workshop on AI methods in fraud and risk management

Li N, Sun H, Chipman KC, George J, Yan X (2014) A probabilistic approach to uncovering attributed graph anomalies. In: Proceedings of the 2014 SIAM international conference on data mining, Philadelphia, pp 82–90, 24–26 Apr 2014. doi:10.1137/1.9781611973440.10, http://dx.doi.org/10.1137/1.9781611973440.10

Lin J, Keogh E, Fu A, Herle HV (2005) Approximations to magic: finding unusual medical time series. In: Proceedings of the 18th IEEE symposium on computer-based medical systems. IEEE Computer

Society, Washington, DC, pp 329–334. doi:http://dx.doi.org/10.1109/CBMS.2005.34

Ma J, Perkins S (2003) Online novelty detection on temporal sequences. In: Proceedings of the 9th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM Press, New York, pp 613–618. doi:http://doi.acm.org/10.1145/956750.956828

Mahoney MV, Chan PK (2002) Learning nonstationary models of normal network tra c for detecting novel attacks. In: Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM Press, pp 376–385. doi:http://doi.acm.org/10.1145/775047.775102

Mahoney MV, Chan PK (2003) Learning rules for anomaly detection of hostile network traffic. In: Proceedings of the 3rd IEEE international conference on data mining. IEEE Computer Society, Los Alamitos, p 601

Mahoney MV, Chan PK (2005) Trajectory boundary modeling of time series for anomaly detection. In: Proceedings of the KDD workshop on data mining methods for anomaly detection, Las Vegas, NV, USA

Mahoney MV, Chan PK, Arshad MH (2003) A machine learning approach to anomaly detection. Technical report CS–2003–06, Department of Computer Science, Florida Institute of Technology Melbourne, FL, 32901

Marchette D (1999) A statistical method for profiling network traffic. In: Proceedings of 1st USENIX workshop on intrusion detection and network monitoring, Santa Clara, pp 119–128

Michael CC, Ghosh A (2000) Two state-based approaches to program-based anomaly detection. In: Proceedings of the 16th annual computer security applications conference, IEEE Computer Society, Los Alamitos, p 21

Müller E, Sanchez PI, Mülle Y, Böhm K (2013) Ranking outlier nodes in subspaces of attributed graphs. In: Workshops proceedings of the 29th IEEE international conference on data engineering. ICDE, pp 216–222

Noble CC, Cook DJ (2003) Graph-based anomaly detection. In: Proceedings of the 9th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM Press, pp 631–636. doi:http://doi.acm.org/10.1145/956750.956831

Otey ME, Ghoting A, Parthasarathy S (2006) Fast distributed outlier detection in mixed-attribute data sets. Data Min Knowl Discov 12(2–3):203. doi:http://dx.doi.org/10.1007/s10618-005-0014-6

Ott L, Pang LX, Ramos FT, Chawla S (2014) On integrated clustering and outlier detection. In: Advances in neural information processing systems, pp 1359–1367

Parra L, Deco G, Miesbach S (1996) Statistical independence and novelty detection with information preserving nonlinear maps. Neural Comput 8 (2):260

Pincombe B (2005) Anomaly detection in time series of graphs using ARMA processes. ASOR Bull 24(4):2

Pokrajac D, Lazarevic A, Latecki LJ (2007) Incremental local outlier detection for data streams. In: Proceedings of IEEE symposium on computational intelligence and data mining

Portnoy L, Eskin E, Stolfo S (2001) Intrusion detection with unlabeled data using clustering. In: Proceedings of ACM workshop on data mining applied to security. citeseer.ist.psu.edu/portnoy01intrusion.html

Protopapas P, Giammarco JM, Faccioli L, Struble MF, Dave R, Alcock C (2006) Finding outlier light curves in catalogues of periodic variable stars. Mon Notices R Astron Soc 369(2):677

Qiao Y, Xin XW, Bin Y, Ge S (2002) Anomaly intrusion detection method based on HMM. Electron Lett 38(13):663

Ramaswamy S, Rastogi R, Shim K (2000) Efficient algorithms for mining outliers from large data sets. In: Proceedings of the 2000 ACM SIGMOD international conference on Management of data. ACM Press, New York, pp 427–438. doi:http://doi.acm.org/10.1145/342009.335437

Roth V (2004) In: NIPS

Rousseeuw PJ, Leroy AM (1987) Robust regression and outlier detection. Wiley, New York

Salvador S, Chan P (2003) Learning states and rules for time-series anomaly detection. Technical report CS–2003–05, Department of Computer Science, Florida Institute of Technology Melbourne FL 32901

Salvador S, Chan P (2005) Learning states and rules for detecting anomalies in time series. Appl Intell 23(3):241. doi:http://dx.doi.org/10.1007/s10489-005-4610-3

Sánchez PI, Müller E, Irmler O, Böhm K (2014) Local context selection for outlier ranking in graphs with multiple numeric node attributes. In: Proceedings of the 26th International conference on scientific and statistical database management (SSDBM '14). ACM, New York, pp 16:1–16:12. doi:10.1145/2618243.2618266. http://doi.acm.org/10.1145/2618243.2618266

Schölkopf B, Platt JC, Shawe-Taylor JC, Smola AJ, Williamson RC (2001) Estimating the support of a high-dimensional distribution. Neural Comput 13(7):1443

Schubert E, Wojdanowski R, Zimek A, Kriegel HP (2012) In: SDM. SIAM/Omnipress, Anaheim, CA, USA, pp 1047–1058

Shekhar S, Lu CT, Zhang P (2001) A novel anomaly detection scheme based on principal component classifier. In: Proceedings of the 7th ACM SIGKDD international conference on knowledge discovery and data mining. ACM Press, New York, pp 371–376. doi:http://doi.acm.org/10.1145/502512.502567

Shyu ML, Chen SC, Sarinnapakorn K, Chang L (2003) A novel anomaly detection scheme based on principal component classifier. In: Proceedings of 3rd

IEEE international conference on data mining, Melbourne, pp 353–365
Song X, Wu M, Jermaine C, Ranka S (2007) Conditional anomaly detection. IEEE Trans Knowl Data Eng 19(5):631 doi:http://doi.ieeecomputersociety.org/10.1109/TKDE.2007.1009
Spence C, Parra L, Sajda P (2001) Detection, synthesis and compression in mammographic image analysis with a hierarchical image probability model. In: Proceedings of the IEEE workshop on mathematical methods in biomedical image analysis. IEEE Computer Society, Washington, DC, p 3
Sun J, Qu H, Chakrabarti D, Faloutsos C (2005) Relevance search and anomaly detection in bipartite graphs. SIGKDD Explor Newslett 7(2):48
Sun P, Chawla S, Arunasalam B (2006) Mining for outliers in sequential databases. In: SIAM international conference on data mining, Philadelphia
Tan PN, Steinbach M, Kumar V (2005) Introduction to data mining. Addison-Wesley, Boston
Tang J, Chen Z, chee Fu AW, Cheung DW (2002) Enhancing effectiveness of outlier detections for low density patterns. In: Proceedings of the Pacific-Asia conference on knowledge discovery and data mining, Taipei, pp 535–548
Tax DMJ (2001) One-class classification; concept-learning in the absence of counter-examples. PhD thesis, Delft University of Technology
Tax D, Duin R (1999) Data domain description using support vectors. In: Verleysen M (ed) Proceedings of the European symposium on artificial neural networks, Brussels, pp 251–256
Tax D, Duin R (1999) Support vector data description. Pattern Recognit Lett 20(11–13):1191
Tong H, Lin C-Y (2011) Non-negative residual matrix factorization with application to graph anomaly detection. In: Proceedings of the 2011 SIAM international conference on data mining, Philadelphia, pp 143–153
Tsay RS, Peja D, Pankratz AE (2000) Outliers in multivariate time series. Biometrika 87(4):789
Vaidya J, Clifton C (2004) Privacy-preserving outlier detection. In: Proceedings of the 4th IEEE international conference on data mining, Brighton, pp 233–240
Wei L, Keogh E, Xi X (2006) Saxually explicit images: Finding unusual shapes. In: Proceedings of the sixth international conference on data mining, IEEE Computer Society, Washington, DC, pp 711–720. doi:http://dx.doi.org/10.1109/ICDM.2006.138
Wei L, Kumar N, Lolla V, Keogh EJ, Lonardi S, Ratanamahatana C (2005) Assumption-free anomaly detection in time series. In: Proceedings of the 17th international conference on Scientific and statistical database management, Lawrence Berkeley Laboratory, Berkeley, pp 237–240
Weigend AS, Mangeas M, Srivastava AN (1995) Nonlinear gated experts for timeseries – discovering regimes and avoiding overfitting. Int J Neural Syst 6(4):373
Wu Q, Shao Z (2005) Network anomaly detection using time series analysis. In: Proceedings of the joint international conference on autonomic and autonomous systems and international conference on networking and services. IEEE Computer Society, Washington, DC, p 42
Yang J, Wang W (2003) CLUSEQ: Efficient and effective sequence clustering. In: Proceedings of international conference on data engineering, Bangalore, pp 101–112
Yankov D, Keogh EJ, Rebbapragada U (2007) Disk aware discord discovery: Finding unusual time series in terabyte sized datasets. In: Proceedings of international conference on data mining, pp 381–390
Ye N (2004) A Markov Chain model of temporal behavior for anomaly detection. In: Proceedings of the 5th annual IEEE information assurance workshop. IEEE, Piscataway
Zare Moayedi H, Masnadi-Shirazi M (2008) ARIMA model for network traffic prediction and anomaly detection. Int Symp Inf Technol 4:1. doi:10.1109/ITSIM.2008.4631947
Zhang J, Wang H (2006) Detecting outlying subspaces for high-dimensional data: the new task, algorithms, and performance. Knowl Inf Syst 10(3):333. doi:http://dx.doi.org/10.1007/s10115-006-0020-z
Zhang X, Fan P, Zhu Z (2003) A new anomaly detection method based on hierarchical HMM. In: Proceedings of the 4th international conference on parallel and distributed computing, applications and technologies, Chengdu, pp 249–252
Zimmermann J, Mohay G (2006) Distributed intrusion detection in clusters based on non-interference. In: ACSW Frontiers ’06: Proceedings of the 2006 Australasian workshops on grid computing and e-research. Australian Computer Society, Darlinghurst, pp 89–95

Ant Colony Optimization

Marco Dorigo and Mauro Birattari
Université Libre de Bruxelles, Brussels, Belgium

Synonyms

ACO

Definition

Ant colony optimization (ACO) is a population-based metaheuristic for the solution of difficult combinatorial optimization problems. In ACO, each individual of the population is an artificial agent that builds incrementally and stochastically a solution to the considered problem. Agents build solutions by moving on a graph-based representation of the problem. At each step their moves define which solution components are added to the solution under construction. A probabilistic model is associated with the graph and is used to bias the agents' choices. The probabilistic model is updated on-line by the agents so as to increase the probability that future agents will build good solutions.

Motivation and Background

Ant colony optimization is so called because of its original inspiration: the foraging behavior of some ant species. In particular, in Beckers et al. (1992) it was demonstrated experimentally that ants are able to find the shortest path between their nest and a food source by collectively exploiting the pheromone they deposit on the ground while walking. Similar to real ants, ACO's artificial agents, also called artificial ants, deposit artificial pheromone on the graph of the problem they are solving. The amount of pheromone each artificial ant deposits is proportional to the quality of the solution the artificial ant has built. These artificial pheromones are used to implement a probabilistic model that is exploited by the artificial ants to make decisions during their solution construction activity.

Structure of the Optimization System

Let us consider a minimization problem $(\mathcal{S}, f)$, where $\mathcal{S}$ is the *set of feasible solutions*, and f is the *objective function*, which assigns to each solution $s \in \mathcal{S}$ a cost value $f(s)$. The goal is to find an optimal solution s^*, that is, a feasible solution of minimum cost. The set of all optimal solutions is denoted by $\mathcal{S}^*$.

Ant colony optimization attempts to solve this minimization problem by repeating the following two steps:

- Candidate solutions are constructed using a parameterized probabilistic model, that is, a parameterized probability distribution over the solution space.
- The candidate solutions are used to modify the model in a way that is intended to bias future sampling toward low cost solutions.

The Ant Colony Optimization Probabilistic Model

We assume that the combinatorial optimization problem $(\mathcal{S}, f)$ is mapped on a problem that can be characterized by the following list of items:

- A finite set $\mathcal{C} = \{c_1, c_2, \ldots, c_{N_c}\}$ of *components*, where N_C is the number of components.
- A finite set $\mathcal{X}$ of *states* of the problem, where a state is a sequence $x = \langle c_i, c_j, \ldots, c_k, \ldots\rangle$ over the elements of $\mathcal{C}$. The length of a sequence x, that is, the number of components in the sequence, is expressed by $|x|$. The maximum length of a sequence is bounded by a positive constant $n < +\infty$.
- A set of (candidate) solutions $\mathcal{S}$, which is a subset of $\mathcal{X}$ (i.e., $\mathcal{S} \subseteq \mathcal{X}$).
- A set of feasible states $\tilde{\mathcal{X}}$, with $\tilde{\mathcal{X}} \subseteq \mathcal{X}$, defined via a set of *constraints* Ω.
- A nonempty set $\mathcal{S}^*$ of optimal solutions, with $\mathcal{S}^* \subseteq \tilde{\mathcal{X}}$ and $\mathcal{S}^* \subseteq \mathcal{S}$.

Given the above formulation (Note that, because this formulation is always possible, ACO can in principle be applied to any combinatorial optimization problem.) artificial ants build candidate solutions by performing randomized walks on the completely connected, weighted graph $\mathcal{G} = (\mathcal{C}, \mathcal{L}, \mathcal{T})$, where the vertices are the components $\mathcal{C}$, the set $\mathcal{L}$ fully connects the components $\mathcal{C}$, and $\mathcal{T}$ is a vector of so-called *pheromone trails* τ. Pheromone trails can be associated with components, connections, or both. Here we assume that the pheromone trails are associated with connections, so that $\tau(i, j)$ is the pheromone associated with the connection between components i and j. It is straightforward to extend the algorithm to the other cases. The graph $\mathcal{G}$ is called the *construction graph*.

To construct candidate solutions, each artificial ant is first put on a randomly chosen vertex of the graph. It then performs a randomized walk by moving at each step from vertex to vertex on the graph in such a way that the next vertex is chosen stochastically according to the strength of the pheromone currently on the arcs. While moving from one node to another of the graph $\mathcal{G}$, constraints Ω may be used to prevent ants from building infeasible solutions. Formally, the solution construction behavior of a generic ant can be described as follows:

ANT_SOLUTION_CONSTRUCTION

- For each ant:
  - Select a start node c_1 according to some problem dependent criterion.
  - Set $k = 1$ and $x_k = \langle c_1 \rangle$.
- While $x_k = \langle c_1, c_2, \ldots, c_k \rangle \in \mathcal{X}, x_k \notin \mathcal{S}$, and the set J_{x_k} of components that can be appended to x_k is not empty, select the next node (component) c_{k+1} randomly according to:

$$P_{\mathcal{T}}(c_{k+1} = c | x_k) = \begin{cases} \dfrac{F_{(c_k,c)}(\tau(c_k,c))}{\sum_{(c_k,y)\in J_{x_k}} F_{(c_k,y)(\tau(c_k,y))}} & \text{if } (c_k,c) \in J_{x_k}, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

  where a connection (c_k, y) belongs to J_{x_k} if and only if the sequence $x_{k+1} = \langle c_1, c_2, \ldots, c_k, y \rangle$ satisfies the constraints Ω (that is, $x_{k+1} \in \tilde{\mathcal{X}}$) and $F_{(i,j)}(z)$ is some monotonic function – a common choice being $z^{\alpha}\eta(i,j)^{\beta}$, where α, $\beta > 0$, and $\eta(i,j)$'s are heuristic values measuring the desirability of adding component j after i. If at some stage $x_k \notin \mathcal{S}$ and $J_{x_k} = \emptyset$, that is, the construction process has reached a dead-end, the current state x_k is discarded. However, this situation may be prevented by allowing artificial ants to build infeasible solutions as well. In such a case, an infeasibility penalty term is usually added to the cost function. Nevertheless, in most of the settings in which ACO has been applied, the dead-end situation does not occur.

For certain problems, one may find it useful to use a more general scheme, where F depends on the pheromone values of several "related" connections rather than just a single one. Moreover, instead of the *random-proportional rule* above, different selection schemes, such as the *pseudo-random-proportional rule* (Dorigo and Gambardella 1997), may be used.

The Ant Colony Optimization Pheromone Update

Many different schemes for pheromone update have been proposed within the ACO framework. For an extensive overview, see Dorigo and Stützle (2004). Most pheromone updates can be described using the following generic scheme:

GENERIC_ACO_UPDATE

- $\forall s \in \hat{S}_t, \forall (i,j) \in s : \tau(i,j) \leftarrow \tau(i,j) + Q_F(s|S_1, \ldots, S_t)$
- $\forall (i,j) : \tau(i,j) \leftarrow (1-\rho) \cdot \tau(i,j)$,

where S_i is the sample in the ith iteration, $\rho, 0 \leq \rho < 1$, is the evaporation rate, and $Q_f(s|S_1, \ldots, S_t)$ is some "quality function," which is typically required to be non-increasing with respect to f and is defined over the "reference set" $\hat{S}_t$.

Different ACO algorithms may use different quality functions and reference sets. For example, in the very first ACO algorithm – Ant System (Dorigo et al. 1991, 1996) – the quality function is simply $1/f(s)$ and the reference set $\hat{S}_t = S_t$. In a subsequently proposed scheme, called *iteration best update* (Dorigo and Gambardella 1997), the reference set is a singleton containing the best solution within S_t (if there are several iteration-best solutions, one of them is chosen randomly). For the *global-best update* (Dorigo et al. 1996; Stützle and Hoos 1997), the reference set contains the best among all the iteration-best solutions (and if there are more than one global-best solution, the earliest one is chosen). In Dorigo et al. (1996) an *elitist* strategy was introduced, in which the update is a combination of the previous two.

In case a good lower bound on the optimal solution cost is available, one may use the following quality function (Maniezzo 1999):

$$Q_f(s|S_1,\ldots,S_t) = \tau_0\left(1 - \frac{f(s)-\mathrm{LB}}{\bar{f}-\mathrm{LB}}\right) = \tau_0\frac{\bar{f}-f(s)}{\bar{f}-\mathrm{LB}}, \tag{2}$$

where $\bar{f}$ is the average of the costs of the last k solutions and LB is the lower bound on the optimal solution cost. With this quality function, the solutions are evaluated by comparing their cost to the average cost of the other recent solutions, rather than by using the absolute cost values. In addition, the quality function is automatically scaled based on the proximity of the average cost to the lower bound.

A pheromone update that slightly differs from the generic update described above was used in *ant colony system* (ACS) (Dorigo and Gambardella 1997). There the pheromone is evaporated by the ants online during the solution construction, hence only the pheromone involved in the construction evaporates.

Another modification of the generic update was introduced in $\mathcal{MAX}-\mathcal{MIN}$ Ant System (Stützle and Hoos 1997, 2000), which uses maximum and minimum pheromone trail limits. With this modification, the probability of generating any particular solution is kept above some positive threshold. This helps to prevent search stagnation and premature convergence to suboptimal solutions.

Cross-References

▶ Swarm Intelligence

Recommended Reading

Beckers R, Deneubourg JL, Goss S (1992) Trails and U-turns in the selection of the shortest path by the ant Lasius Niger. J Theor Biol 159:397–415

Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput 1(1):53–66

Dorigo M, Stützle T (2004) Ant colony optimization. MIT Press, Cambridge

Dorigo M, Maniezzo V, Colorni A (1991) Positive feedback as a search strategy. Technical report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan

Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern – Part B 26(1): 29–41

Maniezzo V (1999) Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. INFORMS J Comput 11(4):358–369

Stützle T, Hoos HH (1997) The $\mathcal{MAX}-\mathcal{MIN}$ ant system and local search for the traveling salesman problem. In: Proceedings of the 1997 congress on evolutionary computation – CEC'97. IEEE Press, Piscataway, pp 309–314

Stützle T, Hoos HH (2000) $\mathcal{MAX}-\mathcal{MIN}$ ant system. Future Gener Comput Syst 16(8):889–914

Anytime Algorithm

An *anytime algorithm* is an algorithm whose output increases in quality gradually with increased running time. This is in contrast to algorithms that produce no output at all until they produce full-quality output after a sufficiently long execution time. An example of an algorithm with good anytime performance is ▶ Adaptive Real-Time Dynamic Programming (ARTDP).

AODE

▶ Averaged One-Dependence Estimators

Apprenticeship Learning

▶ Behavioral Cloning

Approximate Dynamic Programming

▶ Value Function Approximation

Apriori Algorithm

Hannu Toivonen
University of Helsinki, Helsinki, Finland

Definition

Apriori algorithm (Agrawal et al. 1996) is a data mining method which outputs all ▶ frequent itemsets and ▶ association rules from given data.

Input: set $\mathcal{I}$ of items, multiset $\mathcal{D}$ of subsets of $\mathcal{I}$, frequency threshold min_fr, and confidence threshold min_conf.

Output: all frequent itemsets and all valid association rules in $\mathcal{D}$

Method:

```
1: level := 1; frequent_sets : = $\emptyset$;
2: candidate_sets : = $\{\{i\}|i \in \mathcal{I}\}$;
3: while candidate_sets $\neq \emptyset$
    3.1: scan data $\mathcal{D}$ to compute frequencies of all sets in candidate_sets;
    3.2: frequent_sets : = frequent_sets $\cup \{C \in$ candidate_sets $|$frequency$(C) \geq min\_fr\}$;
    3.3: level := level + 1;
    3.4: candidate_sets := $\{A \subset \mathcal{I} \,||\, A \,|=$ level and $B \in$ frequent_sets for all $B \subset A, | B |=$ level $- 1\}$;
4: output frequent_sets;
5: for each $F \in$ frequent_sets
    5.1: for each $E \subset F$, $E \neq \emptyset$, $E \neq F$
        5.1.1: if frequency$(F)$/frequency$(E) \geq min\_conf$ then output association rule $E \rightarrow (F \setminus E)$
```

The algorithm finds frequent itemsets (lines 1–4) by a breadth-first, general-to-specific search. It generates and tests candidate itemsets in batches, to reduce the overhead of database access. The search starts with the most general itemset patterns, the singletons, as candidate patterns (line 2). The algorithm then iteratively computes the frequencies of candidates (line 3.1) and saves those that are frequent (line 3.2). The crux of the algorithm is in the candidate generation (line 3.4): on the next level, those itemsets are pruned that have an infrequent subset. Obviously, such itemsets cannot be frequent. This allows Apriori to find all frequent itemset without spending too much time on infrequent itemsets. See ▶ frequent pattern and ▶ constraint-based mining for more details and extensions.

Finally, the algorithm tests all frequent association rules and outputs those that are also confident (lines 5–5.1.1).

Cross-References

▶ Association Rule
▶ Basket Analysis
▶ Constraint-Based Mining
▶ Frequent Itemset
▶ Frequent Pattern

Recommended Reading

Agrawal R, Mannila H, Srikant R, Toivonen H, Verkamo AI (1996) Fast discovery of association rules. In: Fayyad UM, Piatetsky-Shapiro G, Smyth P, Uthurusamy R (eds) Advances in knowledge discovery and data mining. AAAI Press, Menlo Park, pp 307–328

AQ

▶ Rule Learning

Architecture

▶ Topology of a Neural Network

Area Under Curve

Synonyms

AUC

Definition

The *area under curve* (AUC) statistic is an empirical measure of classification performance based on the area under an ROC curve. It evaluates the performance of a scoring classifier on a test set, but ignores the magnitude of the scores and only takes their rank order into account. AUC is expressed on a scale of 0 to 1, where 0 means that all negatives are ranked before all positives, and 1 means that all positives are ranked before all negatives. See ▶ ROC Analysis.

ARL

▶ Average-Reward Reinforcement Learning

ART

▶ Adaptive Resonance Theory

ARTDP

▶ Adaptive Real-Time Dynamic Programming

Artificial Immune Systems

Jon Timmis
University of York, Heslington, North Yorkshire, UK

Synonyms

AIS; Immune computing; Immune-inspired computing; Immunocomputing; Immunological computation

Definition

Artificial immune systems (AIS) have emerged as a computational intelligence approach that shows great promise. Inspired by the complexity of the immune system, computer scientists and engineers have created systems that in some way mimic or capture certain computationally appealing properties of the immune system, with the aim of building more robust and adaptable solutions. AIS have been defined by de Castro and Timmis (2002) as:

- adaptive systems, inspired by theoretical immunology and observed immune functions, principle and models, which are applied to problem solving

AIS are not limited to machine learning systems, there are a wide variety of other areas in which AIS are developed such as optimization, scheduling, fault tolerance, and robotics (Hart and Timmis 2008). Within the context of machine learning, both supervised and unsupervised approaches have been developed. Immune-inspired learning approaches typically develop a memory set of detectors that are capable of classifying unseen data items (in the case of supervised learning) or a memory set of detectors that represent clusters within the data (in the case of unsupervised learning). Both static and dynamic learning systems have been developed.

Motivation and Background

The immune system is a complex system that undertakes a myriad of tasks. The abilities of the immune system have helped to inspire computer scientists to build systems that *mimic*, in some way, various properties of the immune system. This field of research, AIS, has seen the application of immune-inspired algorithms to a wide variety of areas.

The origin of AIS has its roots in the early theoretical immunology work of Farmer, Perelson, and Varela (Farmer et al. 1986; Varela et al. 1988). These works investigated a number of theoretical ▸ immune network models proposed to describe the maintenance of immune memory in the absence of antigen. While controversial from an immunological perspective, these models began to give rise to an interest from the computing community. The most influential people at crossing the divide between computing and immunology in the early days were Bersini and Forrest. It is fair to say that some of the early work by Bersini (1991) was very well rooted in immunology, and this is also true of the early work by Forrest (1994). It was these works that formed the basis of a solid foundation for the area of AIS. In the case of Bersini, he concentrated on the immune network theory, examining how the immune system maintained its memory and how one might build models and algorithms mimicking that property. With regard to Forrest, her work was focused on computer security (in particular, network intrusion detection) and formed the basis of a great deal of further research by the community on the application of immune-inspired techniques to computer security.

At about the same time as Forrest was undertaking her work, other researchers began to investigate the nature of learning in the immune system and how that might by used to create *machine learning* algorithms (Cooke and Hunt 1995). They had the idea that it might be possible to exploit the mechanisms of the immune system (in particular, the immune network) in learning systems, so they set about doing a proof of concept (Cooke and Hunt 1995). Initial results were very encouraging, and they built on their success by applying the immune ideas to the classification of DNA sequences as either promoter or nonpromoter classes: this work was generalized in Timmis and Neal (2001).

Similar work was carried out by de Castro and Von Zuben (2001), who developed algorithms for use in function optimization and data clustering. Work in dynamic unsupervised machine learning algorithms was also undertaken, meeting with success in works such as Neal (2002). In the supervised learning domain, very little happened until the work by Watkins (2005) (later expanded in Watkins 2005) developed an immune-based classifier known as AIRS, and in the dynamic supervised domain, with the work in Secker et al. (2003) being one of a number of successes.

Structure of the Learning System

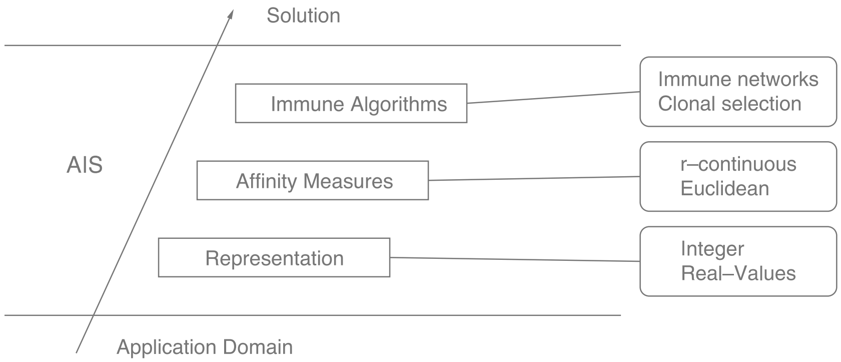
In an attempt to create a common basis for AIS, the work in de Castro and Timmis (2002) proposed the idea of a framework for engineering AIS. They argued that the case for such a framework as the existence of similar frameworks in other biologically inspired approaches, such as ▸ artificial neural networks (ANNs) and evolutionary algorithms (EAs), has helped considerably with the understanding and construction of such systems. For example, de Castro and Timmis (2002) consider a set of artificial neurons,which can be arranged together to form an ANN. In order to acquire knowledge, these neural networks undergo an adaptive process, known as learning or training, which alters (some of) the parameters within the network. Therefore, they argued that in a simplified form, a framework to design an ANN is composed of a set of artificial neurons, a pattern of interconnection for these neurons, and a learning algorithm. Similarly, they argued that in evolutionary algorithms, there is a set of artificial chromosomes representing a population of individuals that iteratively suffer a process of reproduction, genetic variation, and selection. As a result of this process, a population of evolved artificial individuals arises. A framework, in this case, would correspond to the genetic representation of the individuals of the population, plus the procedures for reproduction, genetic variation, and selection. Therefore, they proposed that a framework to design a biologically inspired algorithm requires, at least, the following basic elements:

- A representation for the components of the system
- A set of mechanisms to evaluate the interaction of individuals with the environment and each other. The environment is usually stimulated by a set of input stimuli, one or more fitness function(s), or other means
- Procedures of adaptation that govern the dynamics of the system, i.e., how its behavior varies over time

This framework can be thought of as a layered approach such as the specific framework for engineering AIS of de Castro and Timmis (2002) shown in Fig. 1. This framework follows the three basic elements for designing a biologically inspired algorithm just described, where the set of mechanisms for evaluation are the affinity measures and the procedures of adaptation are the immune algorithms. In order to build a system such as an AIS, one typically requires an application domain or target function. From this basis, the way in which the components of the system will be represented is considered. For example, the representation of network traffic may well be different from the representation of a real-time embedded system. In AIS, the way in which something is represented is known as *shape space*. There are many kinds of shape space, such as Hamming, real valued, and so on, each of which carries it own bias and should be selected with care (Freitas and Timmis 2003). Once the representation has been chosen, one or more affinity measures are used to quantify the interactions of the elements of the system. There are many possible affinity measures (which are partially dependent upon the representation adopted), such as Hamming and Euclidean distance metrics. Again, each of these has its own bias, and the affinity function must be selected with great care, as it can affect the overall performance (and ultimately the result) of the system (Freitas and Timmis 2003).

Supervised Immune-Inspired Learning

The artificial immune recognition system (AIRS) algorithm was introduced as one of the first immune-inspired supervised learning algorithms and has subsequently gone through a period of study and refinement (Watkins 2005). To use classifications from de Castro and Timmis (2002), for the procedures of adaptation, AIRS is a, ▸ clonal selection type of immune-inspired algorithm. The representation and affinity layers



Artificial Immune Systems, Fig. 1 AIS layered framework (Adapted from de Castro and Timmis 2002) of the system are standard in that any number of representations such as binary, real values, etc., can be used with the appropriate affinity function. AIRS has its origin in two other immune-inspired algorithms: CLONALG (CLONAL Selection alGorithm) and Artificial Immune NEtwork (AINE) (de Castro and Timmis 2002). AIRS resembles CLONALG in the sense that both the algorithms are concerned with developing a set of memory cells that give a representation of the learned environment.

AIRS is concerned with the development of a set of memory cells that can encapsulate the training data. This is done in a two-stage process of first evolving a candidate memory cell and then determining if this candidate cell should be added to the overall pool of memory cells. The learning process can be outlined as follows:

1. For each pattern to be recognized, do
   (a) Compare a training instance with all memory cells of the same class and find the memory cell with the best affinity for the training instance. This is referred to as a memory cell $\mathrm{mc}_{\mathrm{match}}$.
   (b) Clone and mutate $\mathrm{mc}_{\mathrm{match}}$ in proportion to its affinity to create a pool of abstract B-cells.
   (c) Calculate the affinity of each B-cell with the training instance.
   (d) Allocate resources to each B-cell based on its affinity.
   (e) Remove the weakest B-cells until the number of resources returns to a preset limit.
   (f) If the average affinity of the surviving B-cells is above a certain level, continue to step 1(g). Else, clone and mutate these surviving B-cells based on their affinity and return to step 1(c).
   (g) Choose the best B-cell as a candidate memory cell ($\mathrm{mc}_{\mathrm{cand}}$).
   (h) If the affinity of $\mathrm{mc}_{\mathrm{cand}}$ for the training instance is better than the affinity of $\mathrm{mc}_{\mathrm{match}}$, then add $\mathrm{mc}_{\mathrm{cand}}$ to the memory cell pool. If, in addition to this, the affinity between $\mathrm{mc}_{\mathrm{cand}}$ and $\mathrm{mc}_{\mathrm{match}}$ is within a certain threshold, then remove $\mathrm{mc}_{\mathrm{match}}$ from the memory cell pool.
2. Repeat from step 1(a) until all training instances have been presented.

Once this training routine is complete, AIRS classifies the instances using k-nearest neighbor with the developed set of memory cells.

Unsupervised Immune-Inspired Learning

The artificial immune network (aiNET) algorithm was introduced as one of the first immune-inspired unsupervised learning algorithms and has subsequently gone through a period of study and refinement (de Castro and Von Zuben 2001). To use classifications from de Castro and Timmis (2002), for the procedures of adaptation, aiNET is an immune network type of immune-inspired algorithm. The representation and affinity layers of the system are standard (the same as in AIRS). aiNET has its origin in another immune-inspired algorithms: CLONALG (the same forerunner to AIRS), and resembles CLONALG in the sense that both algorithms (again) are concerned with developing a set of memory cells that give a representation of the learnt environment. However, within aiNET there is no error feedback into the learning process. The learning process can be outlined as follows:

1. Randomly initialize a population P
2. For each pattern to be recognized, do
   (a) Calculate the affinity of each B-cell (b) in the network for an instance of the pattern being learnt
   (b) Select a number of elements from P into a clonal pool C
   (c) Mutate each element of C proportional to affinity to the pattern being learnt (the higher the affinity, the less mutation applied)
   (d) Select the highest affinity members of C to remain in the set C and remove the remaining elements
   (e) Calculate the affinity between all members of C and remove elements in C that have an affinity below a certain threshold (user defined)

(d) Combine the elements of C with the set P
(e) Introduce a random number of randomly created elements into P to maintain diversity

3. Repeat from 2(a) until stopping criteria is met

Once this training routine is complete, the minimum-spanning tree algorithm is applied to the network to extract the clusters from within the network.

Recommended Reading

Bersini H (1991) Immune network and adaptive control. In: Proceedings of the 1st European conference on artificial life (ECAL). MIT Press, Cambridge, pp 217–226

Cooke D, Hunt J (1995) Recognising promoter sequences using an artificial immune system. In: Proceedings of intelligent systems in molecular biology. AAAI Press, California, pp 89–97

de Castro LN, Timmis J (2002) Artificial immune systems: a new computational intelligence approach. Springer, New York

de Castro LN, Von Zuben FJ (2001) aiNet: an artificial immune network for data analysis. Idea Group Publishing, Hershey, pp 231–259

Farmer JD, Packard NH, Perelson AS (1986) The immune system, adaptation, and machine learning. Physica D 22:187–204

Forrest S, Perelson AS, Allen L, Cherukuri R (1994) Self–nonself discrimination in a computer. In: Proceedings of the IEEE symposium on research security and privacy, Los Alamitos, pp 202–212

Freitas A, Timmis J (2003) Revisiting the foundations of artificial immune systems: a problem oriented perspective. LNCS, vol 2787. Springer, New York, pp 229–241

Hart E, Timmis J (2008) Application areas of AIS: the past, present and the future. J Appl Soft Comput 8(1):191–201

Neal M (2002) An artificial immune system for continuous analysis of time-varying data. In: Timmis J, Bentley P (eds) Proceedings of the 1st international conference on artificial immune system (ICARIS). University of Kent Printing Unit, Canterbury, pp 76–85

Secker A, Freitas A, Timmis J (2003) AISEC: an artificial immune system for email classification. In: Proceedings of congress on evolutionary computation (CEC), Canberra, pp 131–139

Timmis J, Bentley (eds) (2002) Proceedings of the 1st international conference on artificial immune system (ICARIS). University of Kent Printing Unit, Canterbury

Timmis J, Neal M (2001) A resource limited artificial immune system for data analysis. Knowl Based Syst 14(3–4):121–130

Varela F, Coutinho A, Dupire B, Vaz N (1988) Cognitive networks: immune, neural and otherwise. J Theor Immunol 2:359–375

Watkins A (2001) AIRS: a resource limited artificial immune classifier. Master's thesis, Mississippi State University

Watkins A (2005) Exploiting immunological metaphors in the development of serial, parallel and distributed learning algorithms. Ph.D. thesis, University of Kent

Artificial Life

Artificial Life is an interdisciplinary research area trying to reveal and understand the principles and organization of living systems. Its main goal is to artificially synthesize life-like behavior from scratch in computers or other artificial media. Important topics in artificial life include the origin of life, growth and development, evolutionary and ecological dynamics, adaptive autonomous robots, emergence and self-organization, social organization, and cultural evolution.

Artificial Neural Networks

(ANNs) is a computational model based on biological neural networks. It consists of an interconnected group of artificial neurons and processes information using a connectionist approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase.

Cross-References

- ▶ Adaptive Resonance Theory
- ▶ Backpropagation
- ▶ Biological Learning: Synaptic Plasticity, Hebb Rule and Spike Timing Dependent Plasticity
- ▶ Boltzmann Machines

▶ Cascade Correlation
▶ Competitive Learning
▶ Deep Belief Networks
▶ Evolving Neural Networks
▶ Hypothesis Language
▶ Topology of a Neural Network
▶ Neuroevolution
▶ Radial Basis Function Networks
▶ Reservoir Computing
▶ Self-Organizing Maps
▶ Simple Recurrent Network
▶ Weight

Artificial Societies

Jürgen Branke
University of Warwick, Coventry, UK

Synonyms

Agent-based computational models; Agent-based modeling and simulation; Agent-based simulation models

Definition

An artificial society is an agent-based, computer-implemented simulation model of a society or group of people, usually restricted to their interaction in a particular situation. Artificial societies are used in economics and social sciences to explain, understand, and analyze socioeconomic phenomena. They provide scientists with a fully controllable virtual laboratory to test hypotheses and observe complex system behavior emerging as result of the ▶ agents' interaction. They allow formalizing and testing social theories by using computer code, and make it possible to use experimental methods with social phenomena, or at least with their computer representations, on a large scale. Because the designer is free to choose any desired ▶ agent behavior as long as it can be implemented, research based on artificial societies is not restricted by assumptions typical in classical economics, such as homogeneity and full rationality of agents. Overall, artificial societies have added an all new dimension to research in economics and social sciences and have resulted in a new research field called "agent-based computational economics."

Artificial societies should be distinguished from virtual worlds and ▶ artificial life. The term virtual world is usually used for virtual environments to interact with, as, e.g., in computer games. In artificial life, the goal is more to learn about biological principles, understand how life could emerge, and create life within a computer.

Motivation and Background

Classical economics can be roughly divided into analytical and empirical approaches. The former uses deduction to derive theorems from assumptions. Thereby, analytical models usually include a number of simplifying assumptions in order to keep the model tractable, the most typical being full rationality and homogeneity of agents. Also, analytical economics is often limited to equilibrium calculations. Classical empirical economics collects data from the real world, and derives patterns and regularities inductively. In recent years, the tremendous increase in available computational power gave rise to a new branch of economics and sociology which uses simulation of artificial societies as a tool to generate new insights.

Artificial societies are agent-based, computer-implemented simulation models of real societies or a group of people in a specific situation. They are built from the bottom up, by specifying the behavior of the agents in different situations. The simulation then reveals the emerging global behavior of the system, and thus provides a link between micro-level behavior of the agents and macro-level characteristics of the system. Using simulation, researchers can now carry out social experiments under fully controlled and reproducible laboratory conditions, trying out different configurations and observing the consequences.

Like deduction, simulation models are based on a set of clearly specified assumptions as written down in a computer program. This is then used to generate data, from which regularities and patterns are derived inductively. As such, research based on artificial societies stands somewhere between the classical analytical and empirical social sciences.

One of the main advantages of artificial societies is that they allow to consider very complex scenarios where agents are heterogeneous, boundedly rational, or have the ability to learn. Also, they allow to observe evolution over time, instead of just the equilibrium.

Artificial societies can be used for many purposes, e.g.:

1. Verification: Test a hypothesis or theory by examining its validity in relevant, clearly defined scenarios.
2. Explanation: Construct an artificial society which shows the same behavior as the real society. Then analyze the model to explain the emergent behavior.
3. Prediction: Run a model of an existing society into the future. Also, feed the model with different input parameters and use the result as a prediction on how the society would react.
4. Optimization: Test different strategies in the simulation environment, trying to find a best possible strategy.
5. Existence proof: Demonstrate that a specific simulation model is able to generate a certain global behavior.
6. Discovery: Play around with parameter settings, discovering new interdependencies and gaining new insights.
7. Training and education: Use simulation as demonstrator.

Structure of the Learning System

Using artificial societies requires the usual steps in model building and experimental science, including

1. Developing a conceptual model
2. Building the simulation model
3. Verification (making sure the model is correct)
4. Validation (making sure the model is suitable to answer the posed questions)
5. Simulation and analysis using an appropriate experimental design.

Artificial society is an interdisciplinary research area involving, among others, computer science, psychology, economics, sociology, and biology.

Important Aspects

The modeling, simulation, and analysis process described in the previous section is rather complex and only remotely connected to machine learning. Thus, instead of a detailed description of all steps, the following focuses on aspects particularly interesting from a machine learning point of view.

Modeling Learning

One of the main advantages of artificial societies is that they can account for boundedly rational and learning agents. For that, one has to specify (in form of a program) exactly how agents decide and learn.

In principle, all the learning algorithms developed in machine learning could be used, and many have been used successfully, including ▶ reinforcement learning, ▶ artificial neural networks, and ▶ evolutionary algorithms. However, note that the choice of a learning algorithm is not determined by its learning speed and efficiency (as usual in machine learning), but by how well it reflects human learning in the considered scenario, at least if the goal is to construct an artificial society which allows conclusions to be transferred to the real world. As a consequence, many learning models used in artificial societies are motivated by psychology. The idea of the most suitable model depends on the simulation context, e.g., on whether the simulated learning process is conscious or nonconscious, or on the time and effort an individual may be expected to spend on a particular decision.

Besides individual learning (i.e., learning from own past experience), artificial societies usually feature social learning (where one agent learns by observing others), and cultural learning (e.g., the evolution of norms). While the latter simply emerges from the interaction of the agents, the former has to be modeled explicitly. Several different models for learning in artificial societies are discussed in Brenner (2006).

One popular learning paradigm which can be used as a model for individual as well as social learning are ▸ evolutionary algorithms (EAs). Several studies suggest that EAs are indeed an appropriate model for learning in artificial societies, either based on comparisons of simulations with human subject experiments or based on comparisons with other learning mechanisms such as reinforcement learning (Duffy 2006). As EAs are successful search strategies, they seem particularly suitable if the space of possible actions or strategies is very large.

If used to model individual learning, each agent uses a separate EA to search for a better personal solution. In this case, the EA population represents the different alternative actions or strategies that an agent considers. The genetic operators crossover and mutation are clearly related to two major ingredients of human innovation: combination and variation. Crossover can be seen as deriving a new concept by combining two known concepts, and mutation corresponds to a small variation of an existing concept. So, the agent, in some sense, creatively tries out new possibilities. Selection, which favors the best solutions found so far, models the learning part. A solution's quality is usually assessed by evaluating it in a simulation assuming all other agents keep their behavior.

For modeling social learning, EAs can be used in two different ways. In both cases, the population represents the actions or strategies of the different agents in the population. From this it follows that the population size corresponds to the number of agents in the simulation. Fitness values are calculated by running the simulation and observing how the different agents perform. Crossover is now seen as a model for information exchange, or imitation, among agents. Mutation, as in the individual learning case, is seen as a small variation of an existing concept.

The first social learning model simply uses a standard EA, i.e., selection chooses agents to "reproduce," and the resulting new agent strategy replaces an old strategy in the population. While allowing to use standard EA libraries, this approach does not provide a direct link between agents in the simulation and individuals in the EA population. In the second social learning model, each agent directly corresponds to an individual in the EA. In every iteration, each agent creates and tests a new strategy as follows. First, it selects a "donor" individual, with preference to successful individuals. Then it performs a crossover of its own strategy and the donor's strategy, and mutates the result. This can be regarded as an agent observing other agents, and partially adopting the strategies of successful other agents. Then, the resulting new strategy is tested in a "thought experiment," by testing whether the agent would be better off with the new strategy compared with its current strategy, assuming all other agents keep their behavior. If the new strategy performs better, it replaces the current strategy in the next iteration. Otherwise, the new strategy is discarded and the agent again uses its old strategy in the next iteration. The testing of new strategies against their parents has been termed election operator in Arifovic (1994), and makes sure that some very bad and obviously implausible agent strategies never enter the artificial society.

Examples

One of the first forerunners of artificial societies was Schelling's segregation model, 1969. In this study, Schelling placed some artificial agents of two different colors on a simple grid. Each agent follows a simple rule: if less than a given percentage of agents in the neighborhood had the same color, the agent moves to a random free spot. Otherwise, it stays. As the simulation shows, in this model, segregation of agent colors could be observed even if every individual agent was satisfied to live in a neighborhood with only 50 % of its neighbors being of the same color. Thus, with this simple model, Schelling demonstrated that segregation of races in suburbs can occur even if each individual would be happy to live in a diverse neighborhood. Note that the simulations were actually not implemented on a computer but carried out by moving coins on a grid by hand.

Other milestones in artificial societies are certainly the work by Epstein and Axtell on their "sugarscape" model (Epstein and Axtell 1996), and the Santa Fe artificial stock market (Arthur et al. 1997). In the former, agents populate a simple grid world, with sugar growing as the only resource. The agents need the sugar for survival, and can move around to collect it. Axtell and Epstein have shown that even with agents following some very simple rules, the emerging behavior of the overall system can be quite complex and similar in many aspects to observations in the real world, e.g., showing a similar wealth distribution or population trajectories.

The latter is a simple model of a stock market with only a single stock and a risk-free fixed-interest alternative. This model has subsequently been refined and studied by many researchers. One remarkable result of the first model was to demonstrate that technical trading can actually be a viable strategy, something widely accepted in practice, but which classical analytical economics struggled to explain.

One of the most sophisticated artificial societies is perhaps the model of the Anasazi tribe, who left their dwellings in the Long House Valley in northeastern Arizona for so far unknown reasons around 1300 BC (Axtell et al. 2002). By building an artificial society of this tribe and the natural surroundings (climate etc.), it was possible to replicate macro behavior which is known to have occurred and provide a possible explanation for the sudden move.

The NewTies project (Gilbert et al. 2006) has a different and quite ambitious focus: it constructs artificial societies with the hope of an emerging artificial language and culture, which then might be studied to help explain how language and culture formed in human societies.

Software Systems

Agent-based simulations can be facilitated by using specialized software libraries such as Ascape, Netlogo, Repast, StarLogo, Mason, and Swarm. A comparison of different libraries can be found in Railsback (2006).

Applications

Artificial societies have many practical applications, from rather simple simulation models to very complex economic decision problems, examples include traffic simulation, market design, evaluation of vaccination programs, evacuation plans, or supermarket layout optimization. See, e.g., Bonabeau (2002) for a discussion of several such applications.

Future Directions, Challenges

The science on artificial societies is still at its infancy, but the field is burgeoning and has already produced some remarkable results. Major challenges lie in the model building, calibration, and validation of the artificial society simulation model. Despite several agent-based modeling toolkits available, there is a lot to be gained by making them more flexible, intuitive, and user-friendly, allowing to construct complex models simply by selecting and combining provided building blocks of agent behavior. ▶ Behavioral Cloning may be a suitable machine learning approach to generate representative agent models.

Cross-References

▶ Artificial Life
▶ Behavioral Cloning
▶ Coevolutionary Learning
▶ Multi-agent Learning

Recommended Reading

Agent-based computational economics, website maintained by Tesfatsion (2009)

Arifovic J (1994) Genetic algorithm learning and the cobweb-model. J Econ Dyn Control 18:3–28

Arthur B, Holland J, LeBaron B, Palmer R, Taylor P (1997) Asset pricing under endogenous expectations in an artificial stock market. In: Arthur B et al. (eds) The economy as an evolving complex system II. Addison-Wesley, Boston, pp 5–44

Axelrod: The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration (Axelrod 1997)

Axelrod R (1997) The complexity of cooperation: agent-based models of competition and collaboration. Princeton University Press, Princeton

Axtell RL, Epstein JM, Dean JS, Gumerman GJ, Swedlund AC, Harburger J et al (2002) Population growth and collapse in a multiagent model of the kayenta anasazi in long house valley. Proc Natl Acad Sci 99:7275–7279

Bonabeau: Agent-based modeling (Bonabeau 2002)

Brenner: Agent learning representation: Advice on modeling economic learning (Brenner 2006)

Bonabeau E (2002) Agent-based modeling: methods and techniques for simulating human systems. Proc Natl Acad Sci 99:7280–7287

Brenner T (2006) Agent learning representation: advice on modelling economic learning. In: Tesfatsion L, Judd KL (eds) Handbook of computational economics, vol 2. North-Holland, Amsterdam, pp 895–947

Duffy J (2006) Agent-based models and human subject experiments. In: Tesfatsion L, Judd KL (eds) Handbook of computational economics, vol 2. North-Holland, Amsterdam, pp 949–1011

Epstein: Generative social science (Epstein 2006)

Epstein JM (2006) Generative social science: studies in agent-based computational modeling. Princeton University Press, Princeton

Epstein JM, Axtell R (1996) Growing artificial societies. Brookings Institution Press, Washington, DC

Gilbert N, den Besten M, Bontovics A, Craenen BGW, Divina F, Eiben AE et al (2006) Emerging artificial societies through learning. J Artif Soc Soc Simul 9(2). http://jasss.soc.surrey.ac.uk/9/2/9.html

Journal of Artificial Societies and Social Simulation (2009)

Railsback SF, Lytinen SL, Jackson SK (2006) Agent-based simulation platforms: review and development recommendations. Simulation, 82(9):609–623

Schelling TC (1969) Dynamic models of segregation. J Math Soc 2:143–186

Tesfatsion and Judd (eds.): Handbook of computational economics (Tesfatsion and Judd 2006)

Tesfatsion L (2009) Website on agent-based computational economics. http://www.econ.iastate.edu/tesfatsi/ace.htm

Tesfatsion L, Judd KL (eds) (2006a) Handbook of computational economics. Elsevier, Amsterdam/Oxford

Tesfatsion L, Judd KL (eds) (2006b) Handbook of computational economics – vol 2: agent-based computational economics. Elsevier, Amsterdam

The Journal of Artificial Societies and Social Simulation. http://jasss.soc.surrey.ac.uk/JASSS.html

Assertion

In ▶ Minimum Message Length, the code or language shared between sender and receiver that is used to describe the model.

Assessment of Model Performance

▶ Model Evaluation

Association Rule

Hannu Toivonen
University of Helsinki, Helsinki, Finland

Definition

Association rules (Agrawal et al. 1993) can be extracted from data sets where each example consists of a set of items. An association rule has the form $X \rightarrow Y$, where X and Y are ▶ itemsets, and the interpretation is that if set X occurs in an example, then set Y is also likely to occur in the example.

Each association rule is usually associated with two statistics measured from the given data set. The *frequency* or *support* of a rule $X \rightarrow Y$, denoted $\mathrm{fr}(X \rightarrow Y)$, is the number (or alternatively the relative frequency) of examples in which $X \cup Y$ occurs. Its *confidence*, in turn, is the observed conditional probability $P(Y|X) = \mathrm{fr}(X \cup Y)/\mathrm{fr}(X)$.

The ▶ Apriori algorithm (Agrawal et al. 1996) finds all association rules, between any sets X and Y, which exceed user-specified support and confidence thresholds. In association rule mining, unlike in most other learning tasks, the result thus is a set of rules concerning different subsets of the feature space.

Association rules were originally motivated by supermarket ▶ basket analysis, but as a domain independent technique they have found applications in numerous fields. Association rule mining is part of the larger field of ▸ frequent itemset or ▸ frequent pattern mining.

Cross-References

▸ Apriori Algorithm
▸ Basket Analysis
▸ Frequent Itemset
▸ Frequent Pattern

Recommended Reading

Agrawal R, Imieliñski T, Swami A (1993) Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD international conference on management of data, Washington, DC. ACM, New York, pp 207–216

Agrawal R, Mannila H, Srikant R, Toivonen H, Verkamo AI (1996) Fast discovery of association rules. In: Fayyad UM, Piatetsky-Shapiro G, Smyth P, Uthurusamy R (eds) Advances in knowledge discovery and data mining. AAAI Press, Menlo Park, pp 307–328

Associative Bandit Problem

▸ Associative Reinforcement Learning

Associative Reinforcement Learning

Alexander L. Strehl
Rütgers University, New Brunswick, NJ, USA

Synonyms

Associative bandit problem; Bandit problem with side information; Bandit problem with side observations; One-step reinforcement learning

Definition

The *associative reinforcement-learning* problem is a specific instance of the ▸ *reinforcement learning* problem whose solution requires *generalization* and *exploration* but not *temporal credit assignment*. In associative reinforcement learning, an action (also called an arm) must be chosen from a fixed set of actions during successive timesteps and from this choice a real-valued reward or payoff results. On each timestep, an input vector is provided that along with the action determines, often probabilistically, the reward. The goal is to maximize the expected long-term reward over a finite or infinite horizon. It is typically assumed that the action choices do not affect the sequence of input vectors. However, even if this assumption is not asserted, learning algorithms are not required to infer or model the relationship between input vectors from one timestep to the next. Requiring a learning algorithm to discover and reason about this underlying process results in the full reinforcement learning problem.

Motivation and Background

The problem of associative reinforcement learning may be viewed as connecting the problems of ▸ supervised learning or ▸ classification, which is more specific, and reinforcement learning, which is more general. Its study is motivated by real-world applications such as choosing which internet advertisements to display based on information about the user or choosing which stock to buy based on current information related to the market. Both problems are distinguished from supervised learning by the absence of labeled training examples to learn from. For instance, in the advertisement problem, the learner is never told which ads would have resulted in the greatest expected reward (in this problem, reward is determined by whether an ad is clicked on or not). In the stock problem, the best choice is never revealed since the choice itself affects the future price of the stocks and therefore the payoff.

The Learning Setting

The learning problem consists of the following core objects:

- An input space $\mathcal{X}$, which is a set of objects (often a subset of the n-dimension Euclidean space $\mathbb{R}^n$).
- A set of actions or arms $\mathcal{A}$, which is often a finite set of size k.
- A distribution D over $\mathcal{X}$. In some cases, D is allowed to be time-dependent and may be denoted D_t on timestep t for $t = 1, 2, \ldots$.

A learning sequence proceeds as follows. During each timestep $t = 1, 2, \ldots$, an input vector $x_t \in \mathcal{X}$ is drawn according to the distribution D and is provided to the algorithm. The algorithm selects an aarm at $a_t \in \mathcal{A}$. This choice may be stochastic and depend on all previous inputs and rewards observed by the algorithm as well as all previous action choices made by the algorithm for timesteps $t = 1, 2, \ldots$. Then, the learner receives a payoff r_t generated according to some unknown stochastic process that depends only on the x_t and a_t. The informal goal is to maximize the expected long-term payoff. Let $\pi : \mathcal{X} \rightarrow \mathcal{A}$ be any policy that maps input vectors to actions. Let

$$V^{\pi}(T) := E\left[\sum_{i=1}^{T} r_i | a_i \right.$$

$$\left. = \pi(x_i) \text{ for } i = 1, 2, \ldots, T\right] \qquad (1)$$

denotes the expected total reward over T steps obtained by choosing arms according to policy π. The expectation is taken over any randomness in the generation of input vectors x_i and rewards r_i. The expected regret of a learning algorithm with respect to policy π is defined as $V^{\pi}(T) - E[\sum_{i=1}^{T} r_i]$ the expected difference between the return from following policy π and the actual obtained return.

Power of Side Information

Wang et al. (2005) studied the associative reinforcement learning problem from a statistical viewpoint. They considered the setting with two action and analyzed the *expected inferior sampling time*, which is the number of times that the lesser action, in terms of expected reward, is selected. The function mapping input vectors to conditional reward distributions belongs to a known parameterized class of functions, with the true parameters being unknown. They show that, under some mild conditions, an algorithm can achieve finite expected inferior sampling time. This demonstrates the power provided by the input vectors (also called *side observations* or *side information*), because such a result is not possible in the standard *multi-armed bandit problem*, which corresponds to the associative reinforcement-learning problem without input vectors x_i. Intuitively, this type of result is possible when the side information can be used to infer the payoff function of the optimal action.

Linear Payoff Functions

In its most general setting, the associative reinforcement learning problem is intractable. One way to rectify this problem is to assume that the payoff function is described by a linear system. For instance, Abe (1999) and Auer (2002) consider a model where during each timestep t, there is a vector $z_{t,i}$ associated with each arm i. The expected payoff of pulling arm i on this timestep is given by $\theta^T z_{t,i}$ where θ is an unknown parameter vector and θ^T denotes the transpose of f. This framework maps to the framework described above by taking $x_t = (z_{t,1}, z_{t,2}, \ldots, z_{t,k})$. They assume a time-dependent distribution D and focus on obtaining bounds on the regret against the optimal policy. Assuming that all rewards lie in the interval [0, 1], the worst possible regret of any learning algorithm is linear. When considering only the number of timesteps T, Auer (2002) shows that a regret (with respect to the optimal policy) of $O(\sqrt{T}\ln(T))$ can be obtained.

PAC Associative Reinforcement Learning

The previously mentioned works analyze the growth rate of the regret of a learning algorithm with respect to the optimal policy. Another way to approach the problem is to allow the learner some number of timesteps of *exploration*. After the exploration trials, the algorithm is required to output a policy. More specifically, given inputs $0 < \epsilon < 1$ and $0 < \delta < 1$, the algorithm is required to output an ϵ-optimal policy with probability at least $1 - \delta$. This type of analysis is based on the work by Valiant (1984), and learning algorithms satisfying the above condition are termed *probably approximately correct* (PAC).

Motivated by the work of Kaelbling (1994) and Fiechter (PAC associative reinforcement learning, unpublished manuscript, 1995), developed a PAC algorithm when the true payoff function can be described by a *decision list* over the action and input vector. Building on both works, Strehl et al. (2006) showed that a class of associative reinforcement learning problems can be solved efficiently, in a PAC sense, when given a learning algorithm for efficiently solving classification problems.

Recommended Reading

Section 6.1 of the survey by Kaelbling, Littman, and Moore (1996) presents a nice overview of several techniques for the associative reinforcement-learning problem, such as CRBP (Ackley, 1990), ARC (Sutton, 1984), and REINFORCE (Williams, 1992)

Abe N, Long PM (1999) Associative reinforcement learning using linear probabilistic concepts. In: Proceedings of the 16th international conference on machine learning, Bled, pp 3–11

Ackley DH, Littman ML (1990) Generalization and scaling in reinforcement learning. In: Advances in neural information processing systems 2. Morgan Kaufmann, San Mateo, pp 550–557

Auer P (2002) Using confidence bounds for exploitation–exploration trade-offs. J Mach Learn Res 3:397–422

Kaelbling LP (1994) Associative reinforcement learning: functions in k-DNF. Mach Learn 15:279–298

Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: a survey. J Artif Intell Res 4:237–285

Strehl AL, Mesterharm C, Littman ML, Hirsh H (2006) Experience-efficient learning in associative bandit problems. In: Proceedings of the 23rd international conference on machine learning (ICML-06), Pittsburgh, pp 889–896

Sutton RS (1984) Temporal credit assignment in reinforcement learning. Doctoral dissertation, University of Massachusetts, Amherst

Valiant LG (1984) A theory of the learnable. Commun ACM 27:1134–1142

Wang C-C, Kulkarni SR, Poor HV (2005) Bandit problems with side observations. IEEE Trans Autom Control 50:3988–3993

Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach Learn 8:229–256

Attribute

Chris Drummond
National Research Council of Canada, Ottawa, ON, Canada

Synonyms

Characteristic; Feature; Property; Trait

Definition

Attributes are properties of things, ways that we, as humans, might describe them. If we were talking about the appearance of our friends, we might describe one of them as "sex female," "hair brown," "height 5 ft 7 in." Linguistically, this is rather terse, but this very terseness has the advantage of limiting ambiguity. The attributes are sex, hair color, and height. For each friend, we could give the appropriate values to go along with each attribute, some examples are shown in Table 1. Attribute-value pairs are a standard way of describing things within the machine learning community. Traditionally, values have come in one of three types: binary, sex has two values; nominal, hair color has many values; real, height has an ordered set of values. Ideally, the attribute-value pairs are sufficient to describe some things accurately and to tell them apart from others. What might be described is very varied, so the attributes themselves will vary widely.

Motivation and Background

For machine learning to be successful, we need a language to describe everyday things that is sufficiently powerful to capture the similarities and differences between them and yet is computationally easy to manage. The idea that a sufficient number of attribute-value pairs would meet this requirement is an intuitive one. It has also been studied extensively in philosophy and psychology, as a way that humans represent things mentally. In the early days of artificial intelligence research, the frame (Minsky 1974) became a common way of representing knowledge. We have, in many ways, inherited this representation, attribute-value pairs sharing much in common with the labeled slots for values used in frames. In addition, the data for many practical problems comes in this form. Popular methods of storing and manipulating data such as relational databases, and less formal structures such as spread sheets, have columns as attributes and cells as values. So, attribute-value pairs are a ubiquitous way of representing data.

Future Directions

The notion of an attribute-value pair is so well entrenched in machine learning that it is difficult to perceive what might replace it. As, in many practical applications, the data comes in this form, this representation will undoubtedly be around for some time. One change that is occurring is the growing complexity of attribute-values. Traditionally, we have used the simple value types, binary, nominal, and real, discussed earlier. But to effectively describe many things, we need to extend this simple language and use more complex values. For example, in ▸ data mining applied to multimedia, more new complex representations abound. Sound and video streams, images, and various properties of them, are just a few examples (Cord et al. 2005; Simoff and Djeraba 2000).

Attribute, Table 1 Some friends

Sex	Hair color	Height
Male	Black	6 ft 2 in.
Female	Brown	5 ft 7 in.
Female	Blond	5 ft 9 in.
Male	Brown	5 ft 10 in.

Perhaps, the most significant change is away from attributes, albeit with complex values, to structural forms where the relationship between things is included. As Quinlan (1996) states "Data may concern objects or observations with arbitrarily complex structure that cannot be captured by the values of a predetermined set of attributes." There is a large and growing community of researchers in ▸ relational learning. This is evidenced by the number, and growing frequency, of recent workshops at the International Conference for Machine Learning (Cord et al. 2005; De Raedt and Kramer 2000; Dietterich et al. 2004; Fern et al. 2006).

Limitations

In philosophy there is the idea of essence, the properties an object must have to be what it is. In machine learning, particularly in practical applications, we get what we are given and have little control in the choice of attributes and their range of values. If domain experts have chosen the attributes, we might hope that they are properties that can be readily ascertained and are relevant to the task at the hand. For example, when describing one of our friends, we would not say Fred is the one with the spleen. It is not only difficult to observe, it is also poor at discriminating between people. Data are collected for many reasons. In medical applications, all sorts of attribute-values would be collected on patients. Most are unlikely to be important to the current task. An important part of learning is ▸ feature extraction, determining which attributes are necessary for learning.

Whether or not attribute-value pairs are an essential representation for the type of learning required in the development, and functioning, of intelligent agents, remains to be seen. Attribute-values readily capture symbolic information, typically at the level of words that humans naturally use. But if our agents need to move around in their environment, recognizing what they encounter, we might need a different nonlinguistic representation. Certainly, other representations based on a much finer granularity of features, and more holistic in nature, have been central to areas such as ▸ neural networks for some time. In research into ▸ dynamic systems, attractors in a sensor space might be more realistic that attribute-values (See chapter on ▸ Classification).

Recommended Reading

Cord M, Dahyot R, Cunningham P, Sziranyi T (eds) (2005) Workshop on machine learning techniques for processing multimedia content. In: Proceedings of the twenty-second international conference on machine learning, Bonn

De Raedt L, Kramer S (eds) (2000) Workshop on attribute-value and relational learning: crossing the boundaries. In: Proceedings of the seventeenth international conference on machine learning, Stanford University, Palo Alto

Dietterich T, Getoor L, Murphy K (eds) (2004) Workshop on statistical relational learning and its connections to other fields. In: Proceedings of the twenty-first international conference on machine learning, Banff

Fern A, Getoor L, Milch B (eds) (2006) Workshop on open problems in statistical relational learning. In: Proceedings of the twenty-fourth international conference on machine learning, Corvalis

Minsky M (1974) A framework for representing knowledge. Technical report, Massachusetts Institute of Technology, Cambridge

Quinlan JR (1996) Learning first-order definitions of functions. J Artif Intell Res 5:139–161

Simoff SJ, Djeraba C (eds) (2000) Workshop on multimedia data mining. In: Proceedings of the sixth international conference on knowledge discovery and data mining, Boston

Attribute Selection

▸ Feature Selection

Attribute-Value Learning

Attribute-value learning refers to any learning task in which the each ▸ Instance is described by the values of some finite set of attributes (see ▸ Attribute). Each of these instances is often represented as a vector of attribute values, each position in the vector corresponding to a unique attribute.

AUC

▸ Area Under Curve

Authority Control

▸ Record Linkage

Autonomous Helicopter Flight Using Reinforcement Learning

Adam Coates[1], Pieter Abbeel[2], and
Andrew Y. Ng[1,3]
[1]Stanford University, Stanford, CA, USA
[2]EECS Department, UC Berkeley, Stanford, CA, USA
[3]Computer Science Department, Stanford University, Stanford, CA, USA

Definition

Helicopter flight is a highly challenging control problem. While it is possible to obtain controllers for simple maneuvers (like hovering) by traditional manual design procedures, this approach is tedious and typically requires many hours of adjustments and flight testing, even for an experienced control engineer. For complex maneuvers, such as aerobatic routines, this approach is likely infeasible. In contrast, ▶ reinforcement learning (RL) algorithms enable faster and more automated design of controllers. Model-based RL algorithms have been used successfully for autonomous helicopter flight for hovering, forward flight, and using apprenticeship learning methods for expert-level aerobatics. In model-based RL, the first one builds a model of the helicopter dynamics and specifies the task using a reward function. Then, given the model and the reward function, the RL algorithm finds a controller that maximizes the expected sum of rewards accumulated over time.

Motivation and Background

Autonomous helicopter flight represents a challenging control problem and is widely regarded as being significantly harder than control of fixed-wing aircraft (see, e.g., Leishman 2000; Seddon 1990). At the same time, helicopters provide unique capabilities such as in-place hover, vertical takeoff and landing, and low-speed maneuvering. These capabilities make helicopter control an important research problem for many practical applications.

Building autonomous flight controllers for helicopters, however, is far from trivial. When done by hand, it can require many hours of tuning by experts with extensive prior knowledge about helicopter dynamics. Meanwhile, the automated development of helicopter controllers has been a major success story for RL methods. Controllers built using RL algorithms have established state-of-the-art performance for both basic flight maneuvers, such as hovering and forward flight (Bagnell and Schneider 2001; Ng et al. 2004b), as well as being among the only successful methods for advanced aerobatic stunts. Autonomous helicopter aerobatics has been successfully tackled using the innovation of "apprenticeship learning," where the algorithm learns by watching a human demonstrator (Abbeel and Ng 2004). These methods have enabled autonomous helicopters to fly aerobatics as well as an expert human pilot and often even better (Coates et al. 2008).

Developing autonomous flight controllers for helicopters is challenging for a number of reasons:

1. Helicopters have unstable, high-dimensional, asymmetric, noisy, nonlinear, non-minimum phase dynamics. As a consequence, all successful helicopter flight controllers (to date) have many parameters. Controllers with 10–100 gains are not atypical. Hand engineering the right setting for each of the parameters is difficult and time consuming, especially since their effects on performance are often highly coupled through the helicopter's complicated dynamics. Moreover, the unstable dynamics, especially in the low-speed flight regime, complicates flight testing.
2. Helicopters are underactuated: their position and orientation are representable using six parameters, but they have only four control inputs. Thus helicopter control requires significant planning and making trade-offs between errors in orientation and errors in desired position.
3. Helicopters have highly complex dynamics: even though we describe the helicopter as having a 12-dimensional state (position, velocity, orientation, and angular velocity), the true dynamics are significantly more complicated. To determine the precise effects of the inputs, one would have to consider the airflow in a large volume around the helicopter, as well as the parasitic coupling between the different inputs, the engine performance, and the non-rigidity of the rotor blades. Highly accurate simulators are thus difficult to create, and controllers developed in simulation must be sufficiently robust that they generalize to the real helicopter in spite of the simulator's imperfections.
4. Sensing capabilities are often poor: for small remotely controlled (RC) helicopters, sensing is limited because the onboard sensors must deal with a large amount of vibration caused by the helicopter blades rotating at about 30 Hz, as well as higher frequency noise from the engine. Although noise at these frequencies (which are well above the roughly

10 Hz at which the helicopter dynamics can be modeled reasonably) might be easily removed by low pass filtering, this introduces latency and damping effects that are detrimental to control performance. As a consequence, helicopter flight controllers have to be robust to noise and/or latency in the state estimates to work well in practice.

Typical Hardware Setup

A typical autonomous helicopter has several basic sensors on board. An inertial measurement unit (IMU) measures angular rates and linear accelerations for each of the helicopter's three axes. A 3-axis magnetometer senses the direction of the Earth's magnetic field, similar to a magnetic compass (Fig. 1).

Attitude-only sensing, as provided by the inertial and magnetic sensors, is insufficient for precise, stable hovering, and slow-speed maneuvers. These maneuvers require that the helicopter maintains relatively tight control over its position error, and hence high-quality position sensing is needed. GPS is often used to determine helicopter position (with carrier-phase GPS units achieving sub-decimeter accuracy), but vision-based solutions have also been employed (Abbeel et al. 2007; Coates et al. 2008; Saripalliz et al. 2003).

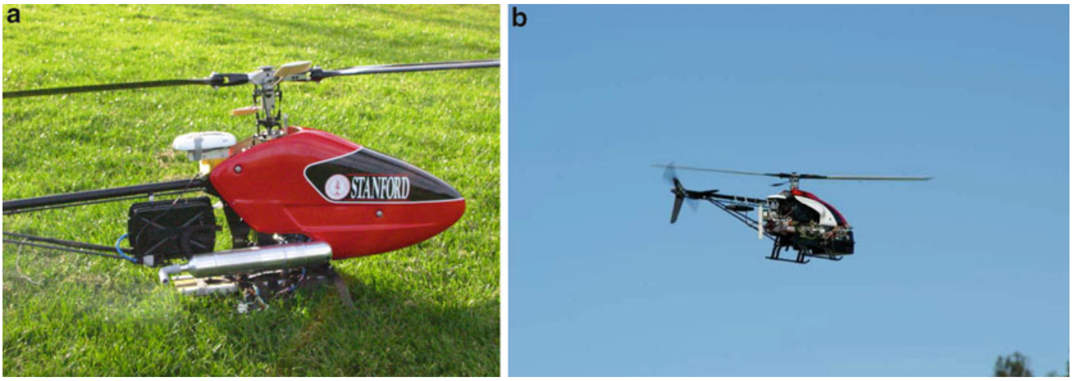
Vibration adds errors to the sensor measurements and may damage the sensors themselves; hence, significant effort may be required to mount the sensors on the airframe (Dunbabin et al. 2004). Provided there is no aliasing, sensor errors added by vibration can be removed by using a digital filter on the measurements (though, again, one must be careful to avoid adding too much latency).

Sensor data from the aircraft sensors is used to estimate the state of the helicopter for use by the control algorithm. This is usually done with an extended Kalman filter (EKF). A unimodal distribution (as computed by the EKF) suffices to represent the uncertainty in the state estimates, and it is common practice to use the mode of the distribution as the state estimate for feedback control. In general the accuracy obtained with this method is sufficiently high that one can treat the state as fully observed.

Most autonomous helicopters have an onboard computer that runs the EKF and the control algorithm (Gavrilets et al. 2002a; La Civita et al. 2006; Ng et al. 2004a). However, it is also possible to use ground-based computers by sending sensor data by wireless to the ground and then transmitting control signals back to the helicopter through the pilot's RC transmitter (Abbeel et al. 2007; Coates et al. 2008).

Helicopter State and Controls

The helicopter state s is defined by its position (p_x, p_y, p_z), orientation (which could be



Autonomous Helicopter Flight Using Reinforcement Learning, Fig. 1 (**a**) Stanford University's instrumented XCell Tempest autonomous helicopter. (**b**) A Bergen Industrial Twin autonomous helicopter with sensors and onboard computer expressed using a unit quaternion q), velocity (v_x, v_y, v_z), and angular velocity $(\omega_x, \omega_y, \omega_z)$.

The helicopter is controlled via a 4-dimensional action space:

1. u_1 and u_2: The lateral (left-right) and longitudinal (front-back) cyclic pitch controls (together referred to as the "cyclic" controls) cause the helicopter to roll left or right and pitch forward or backward, respectively.
2. u_3: The tail rotor pitch control affects tail rotor thrust and can be used to yaw (turn) the helicopter about its vertical axis. In analogy to airplane control, the tail rotor control is commonly referred to as "rudder."
3. u_4: The collective pitch control (often referred to simply as "collective") increases and decreases the pitch of the main rotor blades, thus increasing or decreasing the vertical thrust produced as the blades sweep through the air.

By using the cyclic and rudder controls, the pilot can rotate the helicopter into any orientation. This allows the pilot to direct the thrust of the main rotor in any particular direction, and thus fly in any direction, by rotating the helicopter appropriately.

Helicopter Flight as an RL Problem

Formulation

An RL problem can be described by a tuple $(S, \mathcal{A}, T, H, s(0), R)$, which is referred to as a ▶ Markov decision process (MDP). Here S is the set of states; $\mathcal{A}$ is the set of actions or inputs; T is the dynamics model, which is a set of probability distributions; $\{P^t_{su}\}$ ($P^t_{su}(s'|s,u)$ is the probability of being in state s' at time $t+1$, given the state and action at time t are s and u); H is the horizon or number of time steps of interest; $s(0) \in S$ is the initial state; $R : S \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function.

A policy $\pi = (\mu_0, \mu_1, \ldots, \mu_H)$ is a tuple of mappings from states S to actions $\mathcal{A}$, one mapping for each time $t = 0, \ldots, H$. The expected sum of rewards when acting according to a policy π is given by $U(\pi) = \mathrm{E}[\sum_{t=0}^{H} R(s(t), u(t))|\pi]$. The optimal policy π^* for an MDP $(S, \mathcal{A}, T, H, s(0), R)$ is the policy that maximizes the expected sum of rewards. In particular, the optimal policy is given by: $\pi^* = \arg\max_\pi U(\pi)$.

The common approach to finding a good policy for autonomous helicopter flight proceeds in two steps: First one collects data from manual helicopter flights to build a model. (One could also build a helicopter model by directly measuring physical parameters such as mass, rotor span, etc. However, even when this approach is pursued, one often resorts to collecting flight data to complete the model.) Then one solves the MDP comprised of the model and some chosen reward function. Although the controller obtained, in principle, is only optimal for the learned simulator model, it has been shown in various settings that optimal controllers perform well even when the model has some inaccuracies (see, e.g., Anderson and Moore 1989).

Modeling

One way to create a helicopter model is to use direct knowledge of aerodynamics to derive an explicit mathematical model. This model will depends on a number of parameters that are particular to the helicopter being flown. Many of the parameters may be measured directly (e.g., mass, rotational inertia), while others must be estimated from flight experiments. This approach has been used successfully on several systems (see, e.g., Gavrilets et al. 2001, 2002b; La Civita 2003). However, substantial expert aerodynamics knowledge is required for this modeling approach. Moreover, these models tend to cover only a limited fraction of the flight envelope.

Alternatively, one can learn a model of the dynamics directly from flight data, with only limited a priori knowledge of the helicopter's dynamics. Data is usually collected from a series of manually controlled flights. These flights involve the human sweeping the control sticks back and forth at varying frequencies to cover as much of the flight envelope as possible, while recording the helicopter's state and the pilot inputs at each instant.

Given a corpus of flight data, various different learning algorithms can be used to learn the underlying model of the helicopter dynamics.

If one is only interested in a single flight regime, one could learn a linear model that maps from the current state and action to the next state. Such a model can be easily estimated using ▶ linear regression. (While the methods presented here emphasize time domain estimation, frequency domain estimation is also possible for the special case of estimating linear models Tischler and Cauffman 1992.) Linear models are restricted to small flight regimes (e.g., hover or inverted hover) and do not immediately generalize to full-envelope flight. To cover a broader flight regime, nonparametric algorithms such as locally weighted linear regression have been used (Bagnell and Schneider 2001; Ng et al. 2004b). Nonparametric models that map from current state and action to next state can, in principle, cover the entire flight regime. Unfortunately, one must collect large amounts of data to obtain an accurate model, and the models are often quite slow to evaluate.

An alternative way to increase the expressiveness of the model, without resorting to nonparametric methods, is to consider a time-varying model where the dynamics are explicitly allowed to depend on time. One can then proceed to compute simpler (say, linear) parametric models for each choice of the time parameter. This method is effective when learning a model specific to a trajectory whose dynamics are repeatable but vary as the aircraft travels along the trajectory. Since this method can also require a great deal of data (similar to nonparametric methods) in practice, it is helpful to begin with a non-time-varying parametric model fit from a large amount of data and then augment it with a time-varying component that has fewer parameters (Abbeel et al. 2006; Coates et al. 2008).

One can also take advantage of symmetry in the helicopter dynamics to reduce the amount of data needed to fit a parametric model. Abbeel et al. (2006) observe that – in a coordinate frame attached to the helicopter – the helicopter dynamics are essentially the same for any orientation (or position) once the effect of gravity is removed. They learn a model that predicts (angular and linear) accelerations – except for the effects of gravity – in the helicopter frame as a function of the inputs and the (angular and linear) velocity in the helicopter frame. This leads to a lower-dimensional learning problem, which requires significantly less data. To simulate the helicopter dynamics over time, the predicted accelerations augmented with the effects of gravity are integrated over time to obtain velocity, angular rates, position, and orientation.

Abbeel et al. (2007) used this approach to learn a helicopter model that was later used for autonomous aerobatic helicopter flight maneuvers covering a large part of the flight envelope. Significantly less data is required to learn a model using the gravity-free parameterization compared to a parameterization that directly predicts the next state as a function of current state and actions (as was used in Bagnell and Schneider (2001) and Ng et al. (2004b)). Abbeel et al. evaluate their model by checking its simulation accuracy over longer time scales than just a one-step acceleration prediction. Such an evaluation criterion maps more directly to the reinforcement learning objective of maximizing the expected sum of rewards accumulated over time (see also Abbeel and Ng 2005b).

The models considered above are deterministic. This normally would allow us to drop the expectation when evaluating a policy according to $\mathrm{E}\left[\sum_{t=0}^{H} R(s(t), u(t))|\pi\right]$. However, it is common to add stochasticity to account for unmodeled effects. Abbeel et al. (2007) and Ng et al. (2004a) include additive process noise in their models. Bagnell and Schneider (2001) go further, learning a distribution over models. Their policy must then perform well, on expectation, for a (deterministic) model selected randomly from the distribution.

Control Problem Solution Methods

Given a model of the helicopter, we now seek a policy π that maximizes the expected sum of rewards $U(\pi) = \mathrm{E}\left[\sum_{t=0}^{H} R(s(t), u(t))|\pi\right]$ achieved when acting according to the policy π.

Policy Search

General policy search algorithms can be employed to search for optimal policies for the MDP based on the learned model. Given a policy π, we can directly try to optimize the objective $U(\pi)$. Unfortunately, $U(\pi)$ is an expectation over a complicated distribution making it impractical to evaluate the expectation exactly in general.

One solution is to approximate the expectation $U(\pi)$ by Monte Carlo sampling: under certain boundedness assumptions, the empirical average of the sum of rewards accumulated over time will give a good estimate $\hat{U}(\pi)$ of the expectation $U(\pi)$. Naively applying Monte Carlo sampling to accurately compute, e.g., the local gradient from the difference in function value at nearby points requires very large amounts of samples due to the stochasticity in the function evaluation.

To get around this hurdle, the PEGASUS algorithm (Ng and Jordan 2000) can be used to convert the stochastic optimization problem into a deterministic one. When evaluating by averaging over n simulations, PEGASUS initially fixes n random seeds. For each policy evaluation, the same n random seeds are used so that the simulator is now deterministic. In particular, multiple evaluations of the same policy will result in the same computed reward. A search algorithm can then be applied to the deterministic problem to find an optimum.

The PEGASUS algorithm coupled with a simple local policy search was used by Ng et al. (2004a) to develop a policy for their autonomous helicopter that successfully sustains inverted hover. Bagnell and Schneider (2001) proceed similarly, but use the "amoeba" search algorithm (Nelder and Mead 1964) for policy search.

Because of the searching involved, the policy class must generally have low dimension. Nonetheless, it is often possible to find good policies within these policy classes. The policy class of Ng et al. (2004a), for instance, is a decoupled, linear PD controller with a sparse dependence on the state variables. (For instance, the linear controller for the pitch axis is parametrized as $u_2 = c_0(p_x - p_x^*) + c_1(v_x - v_x^*) + c_2\theta$, which has just three parameters, while the entire state is nine dimensional. Here, $p_\cdot$, $v_\cdot$, and $p_\cdot^*$, $v_\cdot^*$, respectively, are the actual and desired position and velocity. θ denotes the pitch angle.) The sparsity reduces the policy class to just nine parameters. In Bagnell and Schneider (2001), two-layer neural network structures are used with a similar sparse dependence on the state variables. Two neural networks with five parameters each are learned for the cyclic controls.

Differential Dynamic Programming

Abbeel et al. (2007) use differential dynamic programming (DDP) for the task of aerobatic trajectory following. DDP (Jacobson and Mayne 1970) works by iteratively approximating the MDP as linear quadratic regulator (LQR) problems. The LQR control problem is a special class of MDPs, for which the optimal policy can be computed efficiently. In LQR the set of states is given by $S = \mathbb{R}^n$, the set of actions/inputs is given by $\mathcal{A} = \mathbb{R}^p$, and the dynamics model is given by

$$s(t+1) = A(t)s(t) + B(t)u(t) + w(t),$$

where for all $t = 0, \ldots, H$ we have that $A(t) \in \mathbb{R}^{n\times n}$, $B(t) \in \mathbb{R}^{n\times p}$, and $w(t)$ is a mean zero random variable (with finite variance). The reward for being in state $s(t)$ and taking action $u(t)$ is given by

$$-s(t)^\top Q(t)s(t) - u(t)^\top R(t)u(t).$$

Here $Q(t)$, $R(t)$ are positive semi-definite matrices which parameterize the reward function. It is well known that the optimal policy for the LQR control problem is a linear feedback controller which can be efficiently computed using dynamic programming (see, e.g., Anderson and Moore (1989), for details on linear quadratic methods).

DDP approximately solves general continuous state-space MDPs by iterating the following two steps until convergence:

1. Compute a linear approximation to the nonlinear dynamics and a quadratic approximation to the reward function around the trajectory obtained when executing the current policy in simulation.

2. Compute the optimal policy for the LQR problem obtained in Step 1, and set the current policy equal to the optimal policy for the LQR problem.

During the first iteration, the linearizations are performed around the target trajectory for the maneuver, since an initial policy is not available.

This method is used to perform autonomous flips, rolls, and "funnels" (high-speed sideways flight in a circle) in Abbeel et al. (2007) and autonomous autorotation (autorotation is an emergency maneuver that allows a skilled pilot to glide a helicopter to a safe landing in the event of an engine failure or tail-rotor failure) in Abbeel et al. (2008), Fig. 2.

While DDP computes a solution to the nonlinear optimization problem, it relies on the accuracy of the nonlinear model to correctly predict the trajectory that will be flown by the helicopter. This prediction is used in Step 1 above to linearize the dynamics. In practice, the helicopter will often not follow the predicted trajectory closely (due to stochasticity and modeling errors), and thus the linearization will become a highly inaccurate approximation of the nonlinear model. A common solution to this, applied by Coates et al. (2008), is to compute the DDP solution online, linearizing around a trajectory that begins at the current helicopter state. This ensures that the model is always linearized around a trajectory near the helicopter's actual flight path.

Apprenticeship Learning and Inverse RL

In computing a policy for an MDP, simply finding a solution (using any method) that performs well in simulation may not be enough. One may need to adjust both the model and reward function based on the results of flight testing. Modeling error may result in controllers that fly perfectly in simulation but perform poorly or fail entirely in reality. Because helicopter dynamics are difficult to model exactly, this problem is fairly common. Meanwhile, a poor reward function can result in a controller that is not robust to modeling errors or unpredicted perturbations (e.g., it may use large control inputs that are unsafe in practice). If a human "expert" is available to demonstrate the maneuver, this demonstration flight can be leveraged to obtain a better model and reward function.

The reward function encodes both the trajectory that the helicopter should follow and the trade-offs between different types of errors. If the desired trajectory is infeasible (either in the nonlinear simulation or in reality), this results in a significantly more difficult control problem. Also, if the trade-offs are not specified correctly, the helicopter may be unable to compensate for significant deviations from the desired trajectory. For instance, a typical reward function for hovering implicitly specifies a trade-off between position error and orientation error (it is possible to reduce one error, but usually at the cost of increasing the other). If this trade-off is incorrectly

Autonomous Helicopter Flight Using Reinforcement Learning, Fig. 2 Snapshots of an autonomous helicopter performing in-place flips and rolls chosen, the controller may be pushed off course by wind (if it tries too hard to keep the helicopter level) or, conversely, may tilt the helicopter to an unsafe attitude while trying to correct for a large position error.

We can use demonstrations from an expert pilot to recover both a good choice for the desired trajectory and good choices of reward weights for errors relative to this trajectory. In apprenticeship learning, we are given a set of N recorded state and control sequences, $\{s_k(t), u_k(t)\}_{t=0}^{H}$ for $k = 1, \ldots, N$, from demonstration flights by an expert pilot. Coates et al. (2008) note that these demonstrations may be suboptimal but are often suboptimal in different ways. They suggest that a large number of expert demonstrations may implicitly encode the optimal trajectory and propose a generative model that explains the expert demonstrations as stochastic instantiations of an "ideal" trajectory. This is the desired trajectory that the expert has in mind but is unable to demonstrate exactly. Using an Expectation-Maximization (Dempster et al. 1977) algorithm, they infer the desired trajectory and use this as the target trajectory in their reward function.

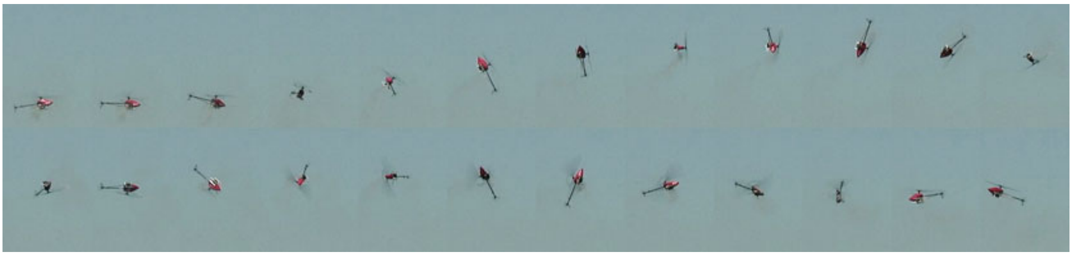
A good choice of reward weights (for errors relative to the desired trajectory) can be recovered using inverse reinforcement learning (Ng and Russell 2000; Abbeel and Ng 2004). Suppose the reward function is written as a linear combination of features as follows: $R(s,u) = c_0\phi_0(s,u) + c_1\phi_1(s,u) + \cdots$. For a single recorded demonstration, $\{s(t), u(t)\}_{t=0}^{H}$, the pilot's accumulated reward corresponding to each feature can be computed as $c_i\phi_i^* = c_i \sum_{t=0}^{H} \phi_i(s(t), u(t))$. If the pilot outperforms the autonomous flight controller with respect to a particular feature ϕ_i, this indicates that the pilot's own "reward function" places a higher value on that feature, and hence its weight c_i should be increased. Using this procedure, a good choice of reward function that makes trade-offs similar to that of a human pilot can be recovered. This method has been used to guide the choice of reward for many maneuvers during flight testing (Abbeel et al. 2007, 2008; Coates et al. 2008).

In addition to learning a better reward function from pilot demonstration, one can also use the pilot demonstration to improve the model directly and attempt to reduce modeling error. Coates et al. (2008), for instance, use errors observed in expert demonstrations to jointly infer an improved dynamics model along with the desired trajectory. Abbeel et al. (2007), however, have proposed the following alternating procedure that is broadly applicable (see also Abbeel and Ng (2005a) for details):

1. Collect data from a human pilot flying the desired maneuvers with the helicopter. Learn a model from the data.
2. Find a controller that works in simulation based on the current model.
3. Test the controller on the helicopter. If it works, we are done. Otherwise, use the data from the test flight to learn a new (improved) model and go back to Step 2.

This procedure has similarities with model-based RL and with the common approach in control to first perform system identification and then find a controller using the resulting model. However, the key insight from Abbeel and Ng (2005a) is that this procedure is guaranteed to converge to expert performance in a polynomial number of iterations. The authors report needing at most three iterations in practice. Importantly, unlike the E^3 family of algorithms (Kearns and Singh 2002), this procedure does not require explicit exploration policies. One only needs to test controllers that try to fly as much as possible (according to the current choice of dynamics model). (Indeed, the E^3-family of algorithms (Kearns and Singh 2002) and its extensions (Kearns and Koller 1999; Brafman and Tennenholtz 2002; Kakade et al. 2003) proceed by generating "exploration" policies, which try to visit inaccurately modeled parts of the state space. Unfortunately, such exploration policies do not even try to fly the helicopter well and thus would almost invariably lead to crashes.)

The apprenticeship learning algorithms described above have been used to fly the most advanced autonomous maneuvers to date. The apprenticeship learning algorithm of Coates et al. (2008), for example, has been used to attain ex-



Autonomous Helicopter Flight Using Reinforcement Learning, Fig. 3 Snapshot sequence of an autonomous helicopter flying a "chaos" maneuver using apprenticeship learning methods. Beginning from the top to left and proceeding left to right and top to bottom, the helicopter performs a flip while pirouetting counterclockwise about its vertical axis (this maneuver has been demonstrated continuously for as long as 15 cycles like the one shown here)

pert level performance on challenging aerobatic maneuvers as well as entire air shows composed of many maneuvers in rapid sequence. These maneuvers include in-place flips and rolls, tic-tocs ("tic-toc" is a maneuver where the helicopter pitches forward and backward with its nose pointed toward the sky (resembling an inverted clock pendulum)), and chaos. ("Chaos" is a maneuver where the helicopter flips in place but does so while continuously pirouetting at a high rate. Visually, the helicopter body appears to tumble chaotically while nevertheless remaining in roughly the same position.) (See Fig. 3.) These maneuvers are considered among the most challenging possible and can only be performed by advanced human pilots. In fact, Coates et al. (2008) show that their learned controller performance can even exceed the performance of the expert pilot providing the demonstrations, putting many of the maneuvers on par with professional pilots (Fig. 4).

A similar approach has been used in Abbeel et al. (2008) to perform the first successful autonomous autorotations. Their aircraft has performed more than 30 autonomous landings successfully without engine power.

Not only do apprenticeship methods achieve state-of-the-art performance, but they are among the fastest learning methods available, as they obviate the need for arduous hand tuning by engineers. Coates et al. (2008), for instance, report that entire air shows can be created from scratch with just 1 h of work. This is in stark contrast to previous approaches that may have required hours or even days of tuning for relatively simple maneuvers.

Conclusion

Helicopter control is a challenging control problem and has recently seen major successes with the application of learning algorithms. This entry has shown how each step of the control design process can be automated using machine learning algorithms for system identification and reinforcement learning algorithms for control. It has also shown how apprenticeship learning algorithms can be employed to achieve expert-level performance on challenging aerobatic maneuvers when an expert pilot can provide demonstrations. Autonomous helicopters with control systems developed using these methods are now capable of flying advanced aerobatic maneuvers (including flips, rolls, tic-tocs, chaos, and autorotation) at the level of expert human pilots.

Cross-References

- ▶ Apprenticeship Learning
- ▶ Reinforcement Learning
- ▶ Reward Shaping

Recommended Reading

Abbeel P, Coates A, Hunter T, Ng AY (2008) Autonomous autorotation of an rc helicopter. In: ISER 11, Athens

Autonomous Helicopter Flight Using Reinforcement Learning, Fig. 4 Superimposed sequence of images of autonomous autorotation landings (from Abbeel et al. 2008)

Abbeel P, Coates A, Quigley M, Ng AY (2007) An application of reinforcement learning to aerobatic helicopter flight. In: NIPS 19, Vancouver, pp 1–8

Abbeel P, Ganapathi V, Ng AY (2006) Learning vehicular dynamics with application to modeling helicopters. In: NIPS 18, Vancouver

Abbeel P, Ng AY (2004) Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the international conference on machine learning, Banff. ACM, New York

Abbeel P, Ng AY (2005a) Exploration and apprenticeship learning in reinforcement learning. In: Proceedings of the international conference on machine learning, Bonn. ACM, New York

Abbeel P, Ng AY (2005b) Learning first order Markov models for control. In: NIPS 18, Vancouver

Abbeel P, Quigley M, Ng AY (2006) Using inaccurate models in reinforcement learning. In: ICML '06: proceedings of the 23rd international conference on machine learning, Pittsburgh. ACM, New York, pp 1–8

Anderson B, Moore J (1989) Optimal control: linear quadratic methods. Prentice-Hall, Princeton

Bagnell J, Schneider J (2001) Autonomous helicopter control using reinforcement learning policy search methods. In: International conference on robotics and automation, Seoul. IEEE, Canada

Brafman RI, Tennenholtz M (2002) R-max, a general polynomial time algorithm for near-optimal reinforcement learning. J Mach Learn Res 3: 213–231

Coates A, Abbeel P, Ng AY (2008) Learning for control from multiple demonstrations. In: Proceedings of the 25th international conference on machine learning (ICML '08), Helsinki

Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J R Stat Soc 39(1):1–38

Dunbabin M, Brosnan S, Roberts J, Corke P (2004) Vibration isolation for autonomous helicopter flight. In: Proceedings of the IEEE international conference on robotics and automation, New Orleans, vol 4, pp 3609–3615

Gavrilets V, Martinos I, Mettler B, Feron E (2002a) Control logic for automated aerobatic flight of miniature helicopter. In: AIAA guidance, navigation and control conference, Monterey. Massachusetts Institute of Technology, Cambridge

Gavrilets V, Martinos I, Mettler B, Feron E (2002b) Flight test and simulation results for an autonomous aerobatic helicopter. In: AIAA/IEEE digital avionics systems conference, Irvine

Gavrilets V, Mettler B, Feron E (2001) Nonlinear model for a small-size acrobatic helicopter. In: AIAA guidance, navigation and control conference, Montreal, pp 1593–1600

Jacobson DH, Mayne DQ (1970) Differential dynamic programming. Elsevier, New York

Kakade S, Kearns M, Langford J (2003) Exploration in metric state spaces. In: Proceedings of the international conference on machine learning, Washington, DC

Kearns M, Koller D (1999) Efficient reinforcement learning in factored MDPs. In: Proceedings of the 16th international joint conference on artificial intelligence, Stockholm. Morgan Kaufmann, San Francisco

Kearns M, Singh S (2002) Near-optimal reinforcement learning in polynomial time. Mach Learn J 49(2–3):209–232

La Civita M (2003) Integrated modeling and robust control for full-envelope flight of robotic helicopters. PhD thesis, Carnegie Mellon University, Pittsburgh

La Civita M, Papageorgiou G, Messner WC, Kanade T (2006) Design and flight testing of a high-bandwidth $\mathcal{H}_\infty$ loop shaping controller for a robotic helicopter. J Guid Control Dyn 29(2):485–494

Leishman J (2000) Principles of helicopter aerodynamics. Cambridge University Press, Cambridge

Nelder JA, Mead R (1964) A simplex method for function minimization. Comput J 7:308–313

Ng AY, Coates A, Diel M, Ganapathi V, Schulte J, Tse B et al (2004) Autonomous inverted helicopter flight via reinforcement learning. In: International symposium on experimental robotics, Singapore. Springer, Berlin

Ng AY, Jordan M (2000) PEGASUS: a policy search method for large MDPs and POMDPs. In: Proceedings of the uncertainty in artificial intelligence 16th conference, Stanford. Morgan Kaufmann, San Francisco

Ng AY, Kim HJ, Jordan M, Sastry S (2004) Autonomous helicopter flight via reinforcement learning. In: NIPS 16, Vancouver

Ng AY, Russell S (2000) Algorithms for inverse reinforcement learning. In: Proceedings of the 17th international conference on machine learning, San Francisco. Morgan Kaufmann, San Francisco, pp 663–670

Saripalli S, Montgomery JF, Sukhatme GS (2003) Visually-guided landing of an unmanned aerial vehicle. IEEE Trans Robot Auton Syst 19(3):371–380

Seddon J (1990) Basic helicopter aerodynamics. AIAA education series. America Institute of Aeronautics and Astronautics, El Segundo

Tischler MB, Cauffman MG (1992) Frequency response method for rotorcraft system identification: flight application to BO-105 couple rotor/fuselage dynamics. J Am Helicopter Soc 37:3–17

Average-Cost Neuro-Dynamic Programming

▶ Average-Reward Reinforcement Learning

Average-Cost Optimization

▶ Average-Reward Reinforcement Learning

Averaged One-Dependence Estimators

Fei Zheng[1,2] and Geoffrey I. Webb[3]
[1]Monash University, Syndey, NSW, Australia
[2]Monash University, Victoria, Australia
[3]Faculty of Information Technology, Monash University, Victoria, Australia

Synonyms

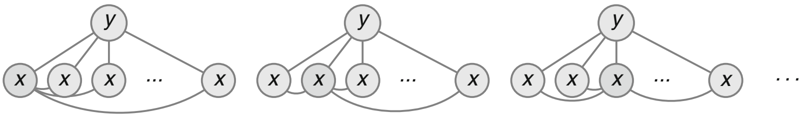
AODE

Definition

Averaged one-dependence estimators is a ▶ semi-naive Bayesian Learning method. It performs classification by aggregating the predictions of multiple one-dependence classifiers in which all attributes depend on the same single parent attribute as well as the class.

Classification with AODE

An effective approach to accommodating violations of naive Bayes' attribute independence assumption is to allow an attribute to depend on other non-class attributes. To maintain efficiency it can be desirable to utilize one-dependence classifiers, such as ▶ Tree Augmented Naive Bayes (TAN), in which each attribute depends upon the class and at most one other attribute. However, most approaches to learning with one-dependence classifiers perform model selection, a process that usually imposes substantial computational overheads and substantially increases variance relative to naive Bayes.



Averaged One-Dependence Estimators, Fig. 1 A Markov network representation of the SPODEs that comprise an example AODE

AODE avoids model selection by averaging the predictions of multiple one-dependence classifiers. In each one-dependence classifier, an attribute is selected as the parent of all the other attributes. This attribute is called the *SuperParent* and this type of one-dependence classifier is called a *SuperParent one-dependence estimator* (SPODE). Only those SPODEs with SuperParent x_i where the value of x_i occurs at least m times are used for predicting a class label y for the test instance $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$. For any attribute value x_i,

$$P(y, \mathbf{x}) = P(y, x_i) P(\mathbf{x} | y, x_i).$$

This equality holds for every x_i. Therefore,

$$P(y, \mathbf{x}) = \frac{\sum_{1 \le i \le n \wedge F(x_i) \ge m} P(y, x_i) P(\mathbf{x} | y, x_i)}{|\{1 \le i \le n \wedge F(x_i) \ge m\}|}, \tag{1}$$

where $F(x_i)$ is the frequency of attribute value x_i in the training sample. Utilizing (1) and the assumption that attributes are independent given the class and the SuperParent x_i, AODE predicts the class for $\mathbf{x}$ by selecting

$$\operatorname*{argmax}_{y} \sum_{1 \le i \le n \wedge F(x_i) \ge m} \hat{P}(y, x_i) \prod_{1 \le j \le n, j \ne i} \hat{P}(x_j | y, x_i). \tag{2}$$

It averages over estimates of the terms in (1), rather than the true values, which has the effect of reducing the variance of these estimates.

Figure 1 shows a Markov network representation of an example AODE.

As AODE makes a weaker attribute conditional independence assumption than naive Bayes while still avoiding model selection, it has substantially lower ▶ bias with a very small increase in variance. A number of studies (Webb et al. 2005; Zheng and Webb 2005) have demonstrated that it often has considerably lower zero-one loss than naive Bayes with moderate time complexity. For comparisons with other semi-naive techniques, see ▶ semi-naive Bayesian learning. One study (Webb et al. 2005) found AODE to provide classification accuracy competitive to a state-of-the-art discriminative algorithm, boosted decision trees.

When a new instance is available, like naive Bayes, AODE only needs to update the probability estimates. Therefore, it is also suited to incremental learning.

In more recent work (Webb et al. 2012), AODE has been generalized to Averaged N-Dependence Estimators (ANDE) and it has been demonstrated that bias can be further decreased by introducing multiple SuperParents to each submodel.

Cross-References

▶ Bayesian Network
▶ Naïve Bayes
▶ Semi-Naive Bayesian Learning
▶ Tree Augmented Naive Bayes

Recommended Reading

Webb GI, Boughton J, Wang Z (2005) Not so naive Bayes: aggregating one-dependence estimators. Mach Learn 58(1):5–24

Webb GI, Boughton J, Zheng F, Ting KM, & Salem H (2012) Learning by extrapolation from marginal to full-multivariate probability distributions: Decreasingly naive Bayesian classification. Mach Learn 86(2): 233–272.

Zheng F, Webb GI (2005) A comparative study of semi-naive Bayes methods in classification learning. In: Proceedings of the fourth Australasian data mining conference, Sydney, pp 141–156

Average-Payoff Reinforcement Learning

▸ Average-Reward Reinforcement Learning

Average-Reward Reinforcement Learning

Prasad Tadepalli
School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA

Synonyms

ARL; Average-cost neuro-dynamic programming; Average-cost optimization; Average-payoff reinforcement learning

Definition

Average-reward reinforcement learning (ARL) refers to learning policies that optimize the average reward per time step by continually taking actions and observing the outcomes including the next state and the immediate reward.

Motivation and Background

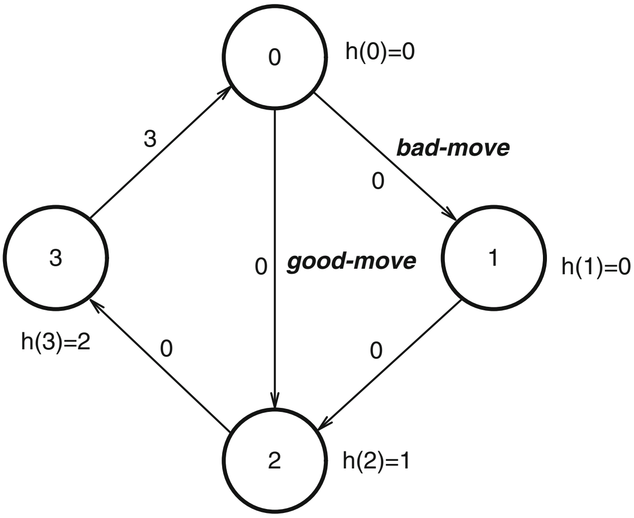
▸ Reinforcement learning (RL) is the study of programs that improve their performance at some task by receiving rewards and punishments from the environment (Sutton and Barto 1998). RL has been quite successful in the automatic learning of good procedures for complex tasks such as playing Backgammon and scheduling elevators (Tesauro 1992; Crites and Barto 1998). In episodic domains in which there is a natural termination condition such as the end of the game in Backgammon, the obvious performance measure to optimize is the expected total reward per episode. But some domains such as elevator scheduling are recurrent, i.e., do not have a natural termination condition. In such cases, total expected reward can be infinite, and we need a different optimization criterion.

In the discounted optimization framework, in each time step, the value of the reward is multiplied by a discount factor $\gamma < 1$, so that the total *discounted* reward is always finite. However, in many domains, there is no natural interpretation for the discount factor γ. A natural performance measure to optimize in such domains is the average reward received per time step. Although one could use a discount factor which is close to 1 to approximate average-reward optimization, an approach that directly optimizes the average reward avoids this additional parameter and often leads to faster convergence in practice.

There is a significant theory behind average-reward optimization based on ▸ Markov decision processes (MDPs) (Puterman 1994). An MDP is described by a 4-tuple $\langle S, A, P, r \rangle$, where S is a discrete set of states and A is a discrete set of actions. P is a conditional probability distribution over the next states, given the current state and action, and r gives the immediate reward for a given state and action. A *policy* π is a mapping from states to actions. Each policy π induces a Markov process over some set of states. In ergodic MDPs, every policy π forms a single closed set of states, and the average reward per time step of π in the limit of infinite horizon is independent of the starting state. We call it the "gain" of the policy π, denoted by $\rho(\pi)$, and consider the problem of finding a "gain-optimal policy," π^*, that maximizes $\rho(\pi)$.

Even though the gain $\rho(\pi)$ of a policy π is independent of the starting state s, the total expected reward in time t is not. It can be denoted by $\rho(\pi)t + h(s)$, where $h(s)$ is a state-dependent bias term. It is the bias values of states that determine which states and actions are preferred and

Average-Reward Reinforcement Learning, Fig. 1 A simple Markov decision process (MDP) that illustrates the Bellman equation



need to be learned for optimal performance. The following theorem gives the Bellman equation for the bias values of states.

Theorem 1 *For ergodic MDPs, there exist a scalar ρ and a real-valued bias function h over S that satisfy the recurrence relation*

$$\forall s \in S, h(s) = \max_{a \in A} \left\{ r(s,a) + \sum_{s' \in S} P(s'|s,a)h(s') \right\} - \rho. \quad (1)$$

Further, the gain-optimal policy μ^* attains the above maximum for each state s, and ρ is its gain.

Note that any one solution to (1) yields an infinite number of solutions by adding the same constant to all h-values. However, all these sets of h-values will result in the same set of optimal policies μ^*, since the optimal action in a state is determined only by the relative differences between the values of h.

For example, in Fig. 1, the agent has to select between the actions good-move and bad-move in state 0. If it stays in state 1, it gets an average reward of 1. If it stays in state 2, it gets an average reward of -1. For this domain, $\rho = 1$ for the optimal policy of choosing good-move in state 0. If we arbitrarily set $h(0)$ to 0, then $h(1) = 0, h(2) = 1$, and $h(3) = 2$ satisfy the recurrence relations in (1). For example, the difference between $h(3)$ and $h(1)$ is 2, which equals the difference between the immediate reward for the optimal action in state 3 and the optimal average reward 1.

Given the probability model P and the immediate rewards r, the above equations can be solved by White's relative value iteration method by setting the h-value of an arbitrarily chosen reference state to 0 and using synchronous successive approximation (Bertsekas 1995). There is also a policy iteration approach to determine the optimal policy starting with some arbitrary policy, solving for its values using the value iteration, and updating the policy using one step look-ahead search. The above iteration is repeated until the policy converges (Puterman 1994).

Model-Based Learning

If the probabilities and the immediate rewards are not known, the system needs to learn them before applying the above methods. A model-based approach called H-learning interleaves model learning with Bellman backups of the value function (Tadepalli and Ok 1998). This is an average-reward version of ▶ Adaptive real-time dynamic programming (Barto et al. 1995). The models are learned by collecting samples of state-action-next-state triples $\langle s, a, s' \rangle$

and computing $P(s'|s,a)$ using the maximum likelihood estimation. It then employs the "certainty equivalence principle" by using the current estimates as the true value while updating the h-value of the current state s according to the following update equation derived from the Bellman equation.

$$h(s) \leftarrow \max_{a \in A} \left\{ r(s,a) + \sum_{s' \in S} P(s'|s,a)h(s') \right\} - \rho. \tag{2}$$

One complication in ARL is the estimation of the average reward ρ in the update equations during learning. One could use the current estimate of the long-term average reward, but it is distorted by the exploratory actions that the agent needs to take to learn about the unexplored parts of the state space. Without the exploratory actions, ARL methods converge to a suboptimal policy. To take this into account, we have from (1), in any state s and a non-exploratory action a that maximizes the right-hand side, $\rho = r(s,a) - h(s) + \sum_{s' \in S} P(s'|S,a)h(s')$. Hence, ρ is estimated by cumulatively averaging $r - h(s) + h(s')$, whenever a greedy action a is executed in state s resulting in state s' and immediate reward r. ρ is updated using the following equation where α is the learning rate.

$$\rho \leftarrow \rho + \alpha(r - h(s) + h(s')). \tag{3}$$

One issue with model-based learning is that the models require too much space and time to learn as tables. In many cases, actions can be represented much more compactly. For example, Tadepalli and Ok (1998) uses dynamic Bayesian networks to represent and learn action models, resulting in significant savings in space and time for learning the models.

Model-Free Learning

One of the disadvantages of the model-based methods is the need to explicitly represent and learn action models. This is completely avoided in model-free methods such as ▶ Q-learning by learning value functions over state–action pairs. Schwartz's R-learning is an adaptation of Q-learning, which is a discounted reinforcement learning method, to optimize average reward (Schwartz 1993).

The state–action value $R(s,a)$ can be defined as the expected long-term advantage of executing action a in state s and from then on following the optimal average-reward policy. It can be defined using the bias values h and the optimal average reward ρ as follows.

$$R(s,a) = r(s,a) + \sum_{s' \in S} P(s'|s,a)h(s') - \rho. \tag{4}$$

The main difference with Q-values is that instead of discounting the expected total reward from the next state, we subtract the average reward ρ in each time step, which is the constant penalty for using up a time step. The h value of any state s can now be defined using the following equation:

$$h(s') = \max_u R(s',u). \tag{5}$$

Initially all the R-values are set to 0. When action a is executed in state s, the value of $R(s,a)$ is updated using the update equation

$$R(s,a) \leftarrow (1-\beta)R(s,a) + \beta(r + h(s') - \rho), \tag{6}$$

where β is the learning rate, r is the immediate reward received, s' is the next state, and ρ is the estimate of the average reward of the current greedy policy. In any state s, the greedy action a maximizes the value $R(s,a)$, so R-learning does not need to explicitly learn the immediate reward function $r(s,a)$ or the action models $P(s'|s,a)$, since it does not use them either for the action selection or for updating the R-values.

Both model-free and model-based ARL methods have been evaluated in several experimental domains (Mahadevan 1996; Tadepalli and Ok 1998). When there is a compact representation for models and can be learned quickly, the model-based method seems to perform better. It also has the advantage of fewer number of tunable parameters. However, model-free methods are more convenient to implement especially if the models are hard to learn or represent.

Scaling Average-Reward Reinforcement Learning

Just as for discounted reinforcement learning, scaling issues are paramount for ARL. Since the number of states is exponential to the number of relevant state variables, a table-based approach does not scale well. The problem is compounded in multi-agent domains where the number of joint actions is exponential in the number of agents. Several function approximation approaches, such as linear functions, multi-layer perceptrons (Marbach et al. 2000), local ▶ linear regression (Tadepalli and Ok 1998), and tile coding (Proper and Tadepalli 2006) were tried with varying degrees of success.

▶ Hierarchical reinforcement learning based on the MAXQ framework was also explored in the average-reward setting and was shown to lead to significantly faster convergence. In the MAXQ framework, we have a directed acyclic graph, where each node represents a task and stores the value function for that task. Usually, the value function for subtasks depends on fewer state variables than the overall value function and hence can be more compactly represented. The relevant variables for each subtask are fixed by the designer of the hierarchy, which makes it much easier to learn the value functions. One potential problem with the hierarchical approach is the loss due to the hierarchical constraint on the policy. Despite this limitation, both model-based (Seri and Tadepalli 2002) and model-free approaches (Ghavamzadeh and Mahadevan 2006) were shown to yield optimal policies in some domains that satisfy the assumptions of these methods.

Applications

A temporal difference method for average reward based on TD(0) was used to solve a call admission control and routing problem (Marbach et al. 2000). On a modestly sized network of 16 nodes, it was shown that the average-reward TD(0) outperforms the discounted version because it required more careful tuning of its parameters. Similar results were obtained in other domains such as automatic guided vehicle routing (Ghavamzadeh and Mahadevan 2006) and transfer line optimization (Wang and Mahadevan 1999).

Convergence Analysis

Unlike their discounted counterparts, both R-learning and H-learning lack convergence guarantees. This is because due to the lack of discounting, the updates can no longer be thought of as contraction mappings, and hence the standard theory of stochastic approximation does not apply. Simultaneous update of the average reward ρ and the value functions makes the analysis of these algorithms much more complicated. However, some ARL algorithms have been proved convergent in the limit using analysis based on ordinary differential equations (ODE) (Abounadi et al. 2002). The main idea is to turn to ordinary differential equations that are closely tracked by the update equations and use two-time-scale analysis to show convergence. In addition to the standard assumptions of stochastic approximation theory, the two-time-scale analysis requires that ρ is updated at a much slower time scale than the value function.

The previous convergence results are based on the limit of infinite exploration. One of the many challenges in reinforcement learning is that of efficient exploration of the MDP to learn the dynamics and the rewards. There are model-based algorithms that guarantee learning an approximately optimal average-reward policy in time polynomial in the numbers of states and actions of the MDP and its mixing time. These algorithms work by alternating between learning the action models of the MDP by taking actions in the environment and solving the learned MDP using offline value iteration.

In the "Explicit Explore and Exploit" or E^3 algorithm, the agent explicitly decides between exploiting the known part of the MDP and optimally trying to reach the unknown part of the MDP (exploration) (Kearns and Singh 2002). During exploration, it uses the idea of "balanced wandering," where the least executed action in the current state is preferred until all actions are executed a certain number of times. In contrast, the R-MAX algorithm implicitly chooses between exploration and exploitation by using the principle of "*optimism under uncertainty*" (Brafman and Tennenholtz 2002). The idea here is to initialize the model parameters optimistically so that all unexplored actions in all states are assumed to reach a fictitious state that yields maximum possible reward from then on regardless of which action is taken. The optimistic initialization of the model parameters automatically encourages the agent to execute unexplored actions, until the true models and values of more states and actions are gradually revealed to the agent. It has been shown that with a probability at least $1 - \delta$, both E^3 and R-MAX learn approximately correct models whose optimal policies have an average reward ϵ-close to the true optimal in time polynomial in the numbers of states and actions, the mixing time of the MDP, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$.

Unfortunately the convergence results do not apply when there is function approximation involved. In the presence of linear function approximation, the average-reward version of temporal difference learning, which learns a state-based value function for a fixed policy, is shown to converge in the limit (Tsitsiklis and Van Roy 1999). The transient behavior of this algorithm is similar to that of the corresponding discounted TD-learning with an appropriately scaled constant basis function (Van Roy and Tsitsiklis 2002). As in the discounted case, development of provably convergent optimal policy learning algorithms with function approximation is a challenging open problem.

Cross-References

- ▶ Efficient Exploration in Reinforcement Learning
- ▶ Hierarchical Reinforcement Learning
- ▶ Model-Based Reinforcement Learning

Recommended Reading

Abounadi J, Bertsekas DP, Borkar V (2002) Stochastic approximation for non-expansive maps: application to Q-learning algorithms. SIAM J Control Optim 41(1):1–22

Barto AG, Bradtke SJ, Singh SP (1995) Learning to act using real-time dynamic programming. Artif Intell 72(1):81–138

Bertsekas DP (1995) Dynamic programming and optimal control. Athena Scientific, Belmont

Brafman RI, Tennenholtz M (2002) R-MAX – a general polynomial time algorithm for near-optimal reinforcement learning. J Mach Learn Res 2:213–231

Crites RH, Barto AG (1998) Elevator group control using multiple reinforcement agents. Mach Learn 33(2/3):235–262

Ghavamzadeh M, Mahadevan S (2006) Hierarchical average reward reinforcement learning. J Mach Learn Res 13(2):197–229

Kearns M, Singh S (2002) Near-optimal reinforcement learning in polynomial time. Mach Learn 49(2/3):209–232

Mahadevan S (1996) Average reward reinforcement learning: foundations, algorithms, and empirical results. Mach Learn 22(1/2/3):159–195

Marbach P, Mihatsch O, Tsitsiklis JN (2000) Call admission control and routing in integrated service networks using neuro-dynamic programming. IEEE J Sel Areas Commun 18(2): 197–208

Proper S, Tadepalli P (2006) Scaling model-based average-reward reinforcement learning for product delivery. In: European conference on machine learning, Berlin. Springer, pp 725–742

Puterman ML (1994) Markov decision processes: discrete dynamic stochastic programming. Wiley, New York

Schwartz A (1993) A reinforcement learning method for maximizing undiscounted rewards. In: Proceedings of the tenth international conference on machine learning, Amherst. Morgan Kaufmann, San Mateo, pp 298–305

Seri S, Tadepalli P (2002) Model-based hierarchical average-reward reinforcement learning. In: Proceedings of international machine learning conference, Sydney. Morgan Kaufmann, pp 562–569

Sutton R, Barto A (1998) Reinforcement learning: an introduction. MIT, Cambridge

Tadepalli P, Ok D (1998) Model-based average-reward reinforcement learning. Artif Intell 100:177–224

Tesauro G (1992) Practical issues in temporal difference learning. Mach Learn 8(3–4):257–277

Tsitsiklis J, Van Roy B (1999) Average cost temporal-difference learning. Automatica 35(11):1799–1808

Van Roy B, Tsitsiklis J (2002) On average versus discounted temporal-difference learning. Mach Learn 49(2/3):179–191

Wang G, Mahadevan S (1999) Hierarchical optimization of policy-coupled semi-Markov decision processes. In: Proceedings of the 16th international conference on machine learning, Bled, pp 464–473

B

Backprop

▶ Backpropagation

Backpropagation

Paul Munro
University of Pittsburgh, Pittsburgh, PA, USA

Synonyms

Backprop; BP; Generalized delta rule

Definition

Backpropagation of error (henceforth *BP*) is a method for training feed-forward neural networks see ▶ Artificial Neural Networks. A specific implementation of BP is an iterative procedure that adjusts network weight parameters according to the gradient of an error measure. The procedure is implemented by computing an error value for each output unit, and by *backpropagating* the error values through the network.

Characteristics

Feed-Forward Networks

A feed-forward neural network is a mathematical function that is composed of constituent "semi-linear" functions constrained by a feed-forward network architecture, wherein the constituent functions correspond to nodes (often called *units* or *artificial neurons*) in a graph, as in Fig. 1. A feed-forward network architecture has a connectivity structure that is an *acyclic graph*; that is, there are no closed loops.

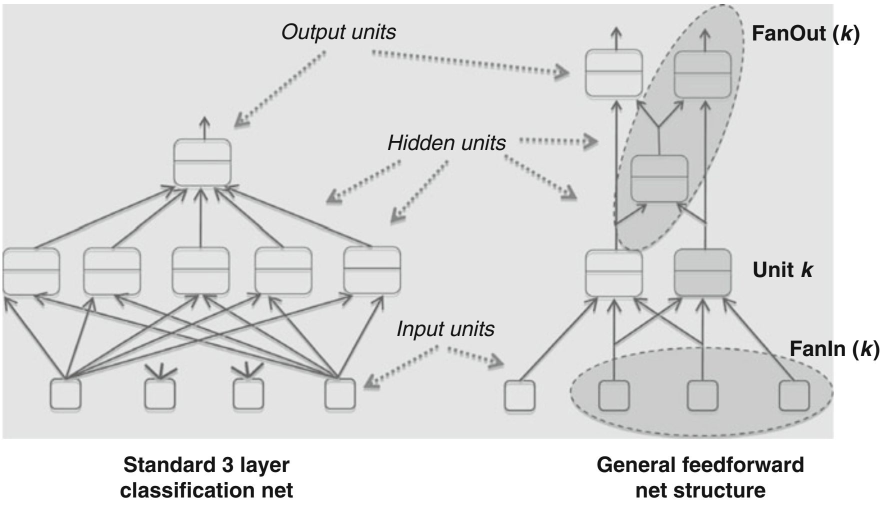
In most cases, the unit functions have a finite range such as [0, 1]. Thus, the network maps $\mathbb{R}^N$ to $[0, 1]^M$, where N is the number of input values and M is the number of output units. Let *FanIn*(k) refer to the set of units that provide input to unit k, and let *FanOut*(k) denote the set of units that receive input from unit k.

In an acyclic graph, at least one unit has a FanIn that is the null set. These are the *input units*; the activity of an input unit is not computed; rather it is set to a value external to the network (i.e., from the training data). Similarly, at least one unit has a null FanOut set. Such units typically represent the output of the network; i.e., this set of values is the result of the network computation. Intermediate units (often called *hidden units*) receive input from other units and project outputs to other computational units.

For the BP procedure, the activity of each unit is computed in two steps:

Linear step: the activities of the FanIn are each multiplied by an independent "weight" parameter, to which a "bias" parameter is added; each computational unit has a single bias parameter, independent of the other units. Let this sum be denoted x_k for unit k.

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1


Output units
Hidden units
Input units
FanOut (*k*)
Unit *k*
FanIn (*k*)
Standard 3 layer classification net
General feedforward net structure

Backpropagation, Fig. 1 Two networks are shown. Input units are shown as simple squares at the bottom of each figure. Other units are computational (designated by a *horizontal line*). *Left*: A standard 3-layer network. Four input units project to five hidden units, which in turn project to a single output unit. Not all connections are shown. Such a network is commonly used for classification tasks. *Right*: An example of a feed-forward network with four inputs, three hidden units, and two outputs

Nonlinear step: The activity a_k of unit k is a differentiable nonlinear function of x_k. A favorite function is the logistic $a = 1/(1 + \exp(-x))$, because it maps the range $[-\infty, +\infty]$ to $[0, 1]$ and its derivative has properties conducive to the implementation of BP.

$$a_k = f_k(x_k); \quad \text{where } x_k = b_k + \sum_{j \varepsilon \text{FanIn}(k)} w_{kj} s_j$$

Gradient Descent

Derivation of BP is a direct application of the *gradient descent* approach to optimization and is dependent on a definition of network error, a function of the actual network response to a stimulus, $\mathbf{r}(\mathbf{s})$ and the target $\mathbf{T}(\mathbf{s})$. The two most common error functions are the summed squared error (SSE) and the cross entropy error (CE) (CE error as defined here is based on the presumption that the output values are in the range [0, 1]. Likewise for the target values; this is often used for classification tasks, wherein target values are set to the endpoints of the range, 0 and 1).

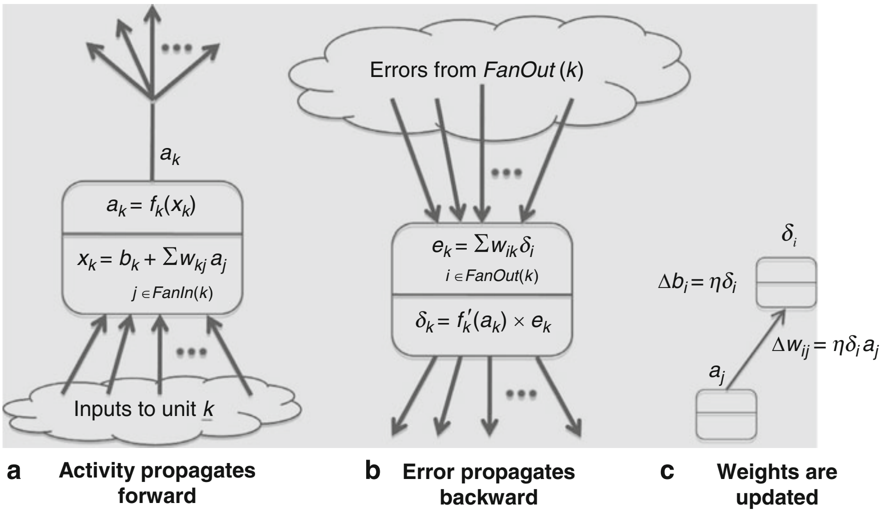
$$E^{SSE} \equiv \sum_{\substack{i \varepsilon \text{Outut} \\ s \varepsilon \text{Train}}} (T_i(\mathbf{s}) - r_i(\mathbf{s}))^2$$

$$E^{CE} \equiv \sum_{\substack{i \varepsilon \text{Outut} \\ s \varepsilon \text{Train}}} [T_i(\mathbf{s}) ln(r_i(\mathbf{s})) - 1 - (T_i(\mathbf{s})) ln(1 - r_i(\mathbf{s}))]$$

Each weight parameter, w_{ij} (the weight of the connection from j to i), is updated by an amount proportional to the negative gradient of the error measure with respect to that parameter:

$$\Delta w_{ij} = -\eta \frac{\partial W}{\partial w_{ij}},$$

where the *step size*, η, modulates the intrinsic tradeoff between smooth convergence of the weights and the speed of convergence; in the regime where η is small, the system is well-behaved and converges smoothly, but slowly, and for larger η, the system may learn some subsets of the training set faster at the expense of smooth convergence on all patterns in the set. Thus, η is also called the *learning rate*.


a_k
$a_k = f_k(x_k)$
$x_k = b_k + \Sigma w_{kj} a_j$
$j \in FanIn(k)$
Inputs to unit k
Errors from *FanOut* (k)
$e_k = \Sigma w_{ik} \delta_i$
$i \in FanOut(k)$
$\delta_k = f'_k(a_k) \times e_k$
δ_i
$\Delta b_i = \eta \delta_i$
$\Delta w_{ij} = \eta \delta_i a_j$
a_j
a Activity propagates forward
b Error propagates backward
c Weights are updated

Backpropagation, Fig. 2 With each iteration of the backprop algorithm, (**a**) An activity value is computed for every unit in the network from the input to the output. (**b**) The network output is compared with the target. The error e_k for output unit k is defined as $(T_k - r_k)$. A value δ_k is computed for each output unit by multiplying e_k by the derivative of the activity function. For hidden units, the error is propagated backward using the weights. (**c**) The weight parameters w_{ij} are updated in proportion to the product of δ_i and a_j

Implementation

Several aspects of the feed-forward network must be defined prior to running a BP program, such as the configuration of the hidden units, the initial values of the weights, the functions they will compute, and the numerical representation of the input and target data. There are also parameters of the learning algorithm that must be chosen, such as the value of η and the form of the error function.

The weight and bias parameters are set to their initial values (these are usually random within specified limits). BP is implemented as an iterative process as follows:

1. A stimulus-target pair is drawn from the training set.
2. The activity values for the units in the network are computed for all the units in the network in a forward fashion from input to output (Fig. 2a).
3. The network output values are compared to the target and a *delta* (δ) value is computed for each output unit based on the difference between the target and the actual output response value.
4. The deltas are propagated backward through the network using the same weights that were used to compute the activity values (Fig. 2b).
5. Each weight is updated by an amount proportional to the product of the downstream delta value and the upstream activity (Fig. 2c).

The procedure can be run either in an *online* mode or *batch* mode. In the online mode, the network parameters are updated for each stimulus-target pair. In the batch mode, the weight changes are computed and accumulated over several iterations without updating the weights until a large number (B) of stimulus-target pairs have been processed (often, the entire training set), at which the weights are updated by the accumulated amounts.

$$\text{online:}\ \ \Delta w_{ij}(t) = \eta\delta_i(t)a_j(t) \quad \Delta b_i(t) = \eta\delta_i(t)$$

$$\text{batch:}\ \ \Delta w_{ij}(t+B) = \sum_{s=t-1}^{t+B} \eta\delta_i(s)a_j(s)$$

$$\Delta b_i(t+T) = \sum_{s=t+1}^{t-B} \eta\delta_i(s)$$

Classification Tasks with BP

The simplest and most common classification function returns a binary value, indicating membership in a particular class. The most common network architecture for a task of this kind is the three-layer network of Fig. 1 (left), with training values of 0 and 1. For classification tasks, the cross entropy error function generally gives significantly faster convergence. After training, the network is in test mode or production mode, and the responses are in the continuous range [0, 1]; the response must thus be interpreted. The value of the response could be interpreted as a probability or fuzzy Boolean value. Often, however, a single threshold is applied to give a binary answer. A double threshold is sometimes used, with the midrange defined as "uncertain."

Curve Fitting with BP

A feed-forward network can be trained to approximate any function, given the sufficient hidden units. The range of the output unit(s) must be capable of generating activity values in the required range. In order to accommodate an arbitrary range uniformly, a linear function is advisable for the output units, and the SSE function is the basis for gradient descent.

The Autoencoder Architecture

The autoencoder is a network design in which the target pattern is identical to the input pattern. The hidden units are configured such that there is a "bottleneck layer" of units that is smaller than the input layer, through which information flows; i.e., there are no connections bypassing the bottleneck. Thus, any information necessary to reconstruct the input pattern at the output layer must be represented at the bottleneck. This approach has been successfully applied as an approach to *nonlinear dimensionality reduction* (e.g., Demers and Cottrell 1993). It bears notable similarities and differences to linear techniques, such as ▸ *principal components analysis* (*PCA*).

Prediction with BP

The plain "vanilla" BP propagates input to output with no explicit representation of time. Several approaches to processing of temporal patterns have been put forward. Most prominent among these are:

Time delay neural network. In this approach, the input stimulus is simply a sample of a time varying signal. The input patterns are typically generated by a sliding window of samples over time or over a sequence.

▸ Simple recurrent network (Elman 1990). A sequence of stimulus patterns is presented as input for the network, which has a single hidden layer design. With each iteration, the input is augmented by a secondary set of input units whose activity is a copy of the hidden layer activity from the previous iteration. Thus, the network is able to maintain a representation of the recent history of network stimuli.

Backpropagation through time (Rumelhart et al. 1986). A recurrent network (i.e., a cyclic network) is "unfolded in time" by forming a large multilayer network, in which each layer is a copy of the entire network shifted in time. Thus, the number of layers limits the temporal window available to the network.

Recurrent backpropagation (Pineda 1989). An acyclic network is run with activity propagation and error propagation, until variables converge. Then the weights are updated.

Cognitive Modeling with BP

Interest in BP as a training technique for classifiers has waned somewhat since the introduction of ▸ *Support vector machines* (SVMs) in the mid 1990s. However, the influence of BP as an approach to modeling cognitive processes, including perception, concept learning, spatial cognition, and language learning, remains strong. Analysis of hidden unit representations (e.g., using clustering techniques) has given insight into plausible intermediate processes that may underlie cognitive phenomena. Also, many cognitive models trained with BP have exhibited time courses consistent with stages of human learning.

Biological Inspiration and Plausibility

The "connectionist" approach to modeling cognition is based on "neural network" models, which have been touted as "biologically inspired" since their inception. The similarities and differences between connectionist architectures and living brains have been exhaustively debated. Like the brain, the models consist of elements that are extremely limited, computationally. Computational power is derived by several units in network architecture. However, there are compelling differences as well. For example, the temporal dynamics in biological neurons is far more complex than the simple functions used in connectionist networks. It remains unclear what level of neurobiological detail is relevant to understand the cognitive functions.

Shortcomings of BP

The BP method is notorious for convergence problems. An inherent problem of gradient descent approaches to optimization is the issue of locally optimal values. Seeking a minimum value be heading downhill is like water running downhill. Not all water reaches the lowest point (sea level). Water that flows into a mountain lake has landed in a local minimum, a region that is bounded by higher ground.

Even when BP converges to a global minimum (or a local minimum that is "good enough"), it is sometimes very slow. The convergence properties of BP depend on the learning rate and random factors, such as the initial weight and bias values.

Another difficulty with BP is the selection of a network structure. The number of hidden units and the interconnectivity among them has a strong influence on both the generalization performance and the convergence time. Since the nature of this influence is poorly understood, the design of the network is left to guesswork. The standard approach is to use a single hidden layer (as in Fig. 1, left), which has the advantage of relatively fast convergence.

History

The idea of training a multilayered network using error propagation was originated by Frank Rosenblatt (1958, 1962). However, he was unable to apply gradient descent because he was using linear threshold functions that were not differentiable; therefore, the technique of gradient descent was unavailable. He developed a technique known as the perceptron learning rule that is only applicable to two layer networks (no hidden units). Without hidden units, the computational power of the network is severely reduced. Work in the field virtually stopped with the publication of *Perceptrons* (Minsky and Papert 1969). The backpropagation procedure was first published by Werbos (1974), but did not receive significant recognition until it was put forward by Rumelhart et al. (1986).

Cross-References

▸ Artificial Neural Networks

Recommended Reading

Demers D, Cottrell G (1993) Non-linear dimensionality reduction. In: Hanson SJ, Cowan JD, Giles CL (eds) Advances in neural information processing systems, vol 5. Morgan Kaufmann, San Mateo

Elman J (1990) Finding structure in time. Cogn Sci 14:179–211

Minsky ML, Papert SA (1969) Perceptrons. MIT Press, Cambridge

Pineda FJ (1989) Recurrent backpropagation and the dynamical approach to adaptive neural computation. Neural Comput 1:161–172

Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65:386–408

Rosenblatt F (1962) Principles of statistical neurodynamics. Spartan, Washington, DC

Werbos P (1974) Beyond regression: new tools for prediction and analysis in the behavioral sciences. Ph.D. thesis, Harvard University, Cambridge

Bagging

Bagging is an ▸ ensemble learning technique. The name "Bagging" is an acronym derived from *B*ootstrap *AGG*regat*ING*. Each member of the ensemble is constructed from a different training dataset. Each dataset is a ▸ bootstrap sample from the original. The models are combined by a uniform average or vote. Bagging works best with ▸ unstable learners, that is those that produce differing generalization patterns with small changes to the training data. Bagging therefore tends not to work well with linear models. See ▶ ensemble learning for more details.

Bake-Off

Definition

Bake-off is a disparaging term for experimental evaluation of multiple learning algorithms by a process of applying each algorithm to a limited set of benchmark problems.

Cross-References

▶ Algorithm Evaluation

Bandit Problem with Side Information

▶ Associative Reinforcement Learning

Bandit Problem with Side Observations

▶ Associative Reinforcement Learning

Basic Lemma

▶ Symmetrization Lemma

Basket Analysis

Hannu Toivonen
University of Helsinki, Helsinki, Finland

Synonyms

Market basket analysis

Definition

The goal of basket analysis is to utilize large volumes of electronic receipts, stored at the checkout terminals of supermarkets, for better understanding of customer behavior.

While many forms of learning and mining can be applied to market baskets, the term usually refers to some variant of ▶ association rule mining. In the basic setting, each market basket constitutes an example essentially defined by the set of purchased products. Association rules then identify sets of items that tend to be bought together. A classical, anecdotal discovery from supermarket data is that "if a basket contains diapers then it often also contains beer." This example illustrates several potential benefits of market basket analysis by association rules: simplicity and understandability of the results, actionability of the results, and a form of nonsupervised approach where the consequent of the rule has not been fixed by the user.

Association rules are often found with the ▶ Apriori algorithm, and are based on ▶ frequent itemsets.

Cross-References

▶ Apriori Algorithm
▶ Association Rule
▶ Frequent Itemset
▶ Frequent Pattern

Batch Learning

Synonyms

Offline Learning

Definition

A *batch learning algorithm* accepts a single input that is a set or sequence of observations. The algorithm produces its model, and does no further learning. Batch learning stands in contrast to ▸ online learning.

Baum-Welch Algorithm

The Baum–Welch algorithm is used for computing maximum likelihood estimates and posterior mode estimates for the parameters (transition and emission probabilities) of a HMM, when given only output sequences (emissions) as training data.

The Baum–Welch algorithm is a particular instantiation of the expectation-maximization algorithm, suited for HMMs.

Bayes Adaptive Markov Decision Processes

▸ Bayesian Reinforcement Learning

Bayes Net

▸ Bayesian Network

Bayes' Rule

Geoffrey I. Webb
Faculty of Information Technology, Monash
University, Victoria, Australia

Synonyms

Bayes' Theorem

Definition

Bayes' rule provides a decomposition of a conditional probability that is frequently used in a family of learning techniques collectively called *Bayesian learning*. Bayes' rule is the equality

$$P(z|w) = \frac{P(z)P(w|z)}{P(w)} \qquad (1)$$

P(w) is called the *prior probability*, P($w|z$) is called the *posterior probability*, and P($z|w$) is called the *likelihood*.

Discussion

Bayes' rule is used for two purposes. The first is *Bayesian update*. In this context z represents some new information that has become available since an estimate P(w) was formed of some hypothesis w. The application of Bayes' rule enables a new estimate of the probability of w (the posterior probability) to be calculated from estimates of the prior probability, the likelihood, and P(z).

The second common application of Bayes' rule is for estimating posterior probabilities in probabilistic learning, where it is the core of Bayesian networks, naïve Bayes, and semi-naïve Bayesian techniques.

While Bayes' rule may initially appear mysterious, it is readily derived from the basic principle of conditional probability that

$$P(w|z) = \frac{P(w,z)}{P(z)} \qquad (2)$$

As

$$P(w,z) = \frac{P(w)P(w,z)}{P(w)} \qquad (3)$$

and

$$\frac{P(w,z)}{P(w)} = P(z|w), \qquad (4)$$

Bayes' rule (Eq. 1) follows by simple substitution of Eq. 4 into Eq. 3 and then of the result into Eq. 2.

Cross-References

▸ Bayesian Network
▸ Naïve Bayes
▸ Semi-naive Bayesian Learning

Bayes' Theorem

▶ Bayes' Rule

Bayesian Methods

Wray L. Buntine
Statistical Machine Learning Program, NICTA, Canberra, ACT, Australia
Faculty of Information Technology, Monash University, Clayton, VIC, Australia

Definition

The two most important concepts used in Bayesian modeling are *probability* and *utility*. Probabilities are used to model our belief about the state of the world and utilities are used to model the *value* to us of different outcomes, thus to model costs and benefits. Probabilities are represented in the form of $p(x|C)$, where C is the current known context and x is some event(s) of interest from a space χ. The left and right arguments of the probability function are in general propositions (in the logical sense). Probabilities are updated based on new evidence or outcomes y using *Bayes rule*, which takes the form

$$p(x|C,y) = \frac{p(x|C)p(Y|x,C)}{p(y|C)},$$

where χ is the discrete domain of x. More generally, any measurable set can be used for the domain χ. An integral or mixed sum and integral can replace the sum. For a utility function $u(x)$ of some event x, for instance the benefit of a particular outcome, the *expected value* of $u()$ is

$$\mathcal{E}_{x|C}[u(x)] = \sum_{x\in\chi} p(x|C)u(x).$$

One then estimates the expected utility $\mathcal{E}_{x|c,y}[u(x)]$ based on different evidence, actions or outcomes y. An action is taken to maximize this expected utility, appealing to the principle of *maximum expected utility (MEU)*. A common application of this principle is recursive: one should take the action now that will maximize utility in the future, assuming all future actions are also taken to maximize utility.

Motivation and Background

In modeling a problem, primarily, one considers an interrelated space of *events* or *states, actions*, and *outcomes*. Events describe the state of the world, outcomes are also sometimes considered events but they are special in that one directly obtains from them costs or benefits. Actions allow one to influence the world. Some actions may instigate tests and thus also help measure the state of the world to reduce uncertainty. Some problems may be dynamic in that a sequence of actions and outcomes are considered and the resulting changes in states modeled.

The Bayesian approach is a modeling methodology that provides a principled approach of how to reason and act in the context of uncertainty and a dynamic environment. In the approach, probabilities are used to model all forms of belief or proportions about events and states, and then utilities are used to model the costs and benefits of any actions taken. An explicit assumption is that these probabilities and utilities can be adequately elicited and precisely modeled for the problem. An implicit assumption is that the computation required – recursive evaluation of possibly nested integrals and sums (over domain variables) – can be done quickly enough so that the computation itself does not become a significant factor in the costs considered.

The Bayesian approach is named after Rev. Thomas Bayes, whose work was contributed to the Royal Society in 1763 after his death, although it was independently more generally presented as a theory by Laplace in 1812. The field was subsequently developed into a field of statistics, inference and decision theory by a stream of authors in the 1900s including Jeffreys (Bernardo and Smith 1994). The field of statistics was dominated by the frequentist school during the 1990s, and for a time Bayesian methods were considered controversial. Like the different schools of theory in machine learning, these statistical approaches now coexist.

The Bayesian approach can be justified by axiomatic prescriptions of how a rational agent should reason and act, and by appeal to principles of consistency. In the context of learning, probabilities are used to infer models of the problem of interest, and then utilities are used to recommend predictions or analysis based on the models.

Theory

Basic Theory

First, consider definitions, the different kinds of probability, the process of reasoning (about probabilities), and making decisions.

Basic definitions: Probabilities are represented in the form of $p(x|C)$, where C is the current known context and x is some event(s) of interest. It is sufficient to place in C only terms relevant to x and ignore terms assumed by default. Moreover, both x and C must have well-defined events. For instance, x = "John is tall" is not considered a well-defined event since the word "tall" is not precise. One would instead replace it with something like x = "John is greater than 6 foot tall" or x = "Julie said John is tall."

An important functional used with probabilities is the *expected value*. For a function $f(x)$ of some event x from a space χ, the expected value of $f()$ is $\mathcal{E}_{x\in\chi[f(x)]}$.

Utility is used to measure value or relative satisfaction, and is usually represented as a function on outcomes. Costs are negative utility and benefits are positive. Utilities should be additive in worth, and are often practically interpreted in monetary units. Strictly speaking, the value of money is nonlinear (for most people, 2 billion dollars is not significantly better than 1 billion dollars), so it is not a correct utility measure. However, it is adequate when the range of financial transactions expected is reasonable.

Expected utility, which is the expected value of the utility function, is the fundamental quantity assessed with Bayesian methods. Some scenarios are the following:

Prediction: For prediction problems, the outcome is the "true" value, and the utility is sometimes the mean square error or the absolute error. In data mining, the choices are much richer, see ► Model Evaluation.

Diagnosis: The outcome is the "true" diagnosis, and utility is made up of the differing costs of treatment, mistreatment, and delay or nontreatment, as well as any benefit from correct diagnosis.

Game playing: The utility comes from the eventual outcome of the game, each player has their own utility and the state of the game constantly changes as plays are made.

In Bayesian machine learning, we usually take utilities as a given, and the majority of the work revolves around evaluating and estimating probabilities and maximizing of expected utility. In some ranking tasks and generalized agent learning, the utilities themselves may be poorly understood.

Belief and proportions: Some probabilities correspond to *proportions* that exist in the real world, such as the proportion of school children in the general population of a given state. These real proportions can be measured by counting or sampling, and they are governed by Kolmogorov's Axioms for probability, including the probability of certainty is 1 and the probability of a disjunction of mutually exclusive events is the sum of the probabilities of the individual events. This kind of probability is used in the *Frequentist School* that only considers long term average proportions obtained from a series of independent and identical experiments. These proportions can be model parameters one wishes to reason about.

Probabilities can also represent beliefs. For instance, in 2000, one could have had a belief about the event that George Bush would win the 2001 Presidential Election in the USA. This event is unique and has only one outcome, so the frequentist notion cannot be justified, i.e., there is no long-term sequence of different 2001 presidential elections with George Bush. Beliefs are usually considered to be *subjective*, in that they are specific to each agent, reflecting their sum of unique experiences, and the unique context in which the event in question occurs.

To better understand the role beliefs play in Bayesian methods, also see ▶ Prior Probability.

Reasoning: A stylized version of probabilistic reasoning considers an event of interest one is reasoning about, x, and evidence, y, one may obtain. Typical scenarios are

Learning: $x = (\boldsymbol{\Theta}, M)$ are parameters $\boldsymbol{\Theta}$ of a model from family M, and $y = \{\boldsymbol{D}\}$ is a set of data $\{\boldsymbol{D}\} = \{d_1, \ldots, d_N\}$. So one considers $p(\boldsymbol{\Theta}, M|\boldsymbol{D}, C)$ versus $p(\boldsymbol{\Theta}, M|C)$.

Diagnosis: x a disease or condition, and y is a set of observable symptoms or diagnostic tests. One might choose a test y that maximizes the expected utility.

Hypothesis testing: x is a hypothesis H and y is some sequence of evidence $E_1, E_2, \ldots, E_n$, so we consider $p(H|E_1, E_2, \ldots, E_n)$ and hope it is sufficiently high.

Different probabilities are then considered:

$p(x|C)$: *The prior probability* for event x, called the base-rate in some contexts.

$p(y|C)$: The *prior probability* for evidence y. Once the evidence has been seen, this is also used as a proxy for the quality of the model.

$p(x|y, C)$: The *posterior probability* for event x given evidence y.

$p(y|x, C)$: The *likelihood* for the event x based on evidence y.

In the case of diagnostic reasoning, the prior $p(x|C)$ is usually the base rate for the disease or condition, and can be got from the population base rate.

In the case of learning, however, the prior $p(\boldsymbol{\Theta}, M|C)$ represents a prior distribution on parameters about which we may well be largely ignorant, or at least may not be able to readily elicit from experts. For instance, the proportion θ_D might be the probability of a new drug slowing the onset of AIDS related diseases. At the moment of initial testing, θ_D is unknown so one places a probability distribution over θ_D, which represents one's belief about the proportion.

These priors are second-order probabilities, beliefs about proportions, and they are the most challenging quantity modeled with the Bayesian approach. They can be a function on thousands of parameters, and can be critical in the success of applications. They are also challenging from the philosophical perspective.

Decision theory: The term *Bayesian inference* is usually reserved for the process of manipulating priors and posteriors, computing probabilities, and computing expected values. *Bayesian decision theory* describes the process of formulating utilities and then evaluating the (sometimes) recursive maximum expected utility formula, such as in game playing, or interactive advertising.

In Bayesian theory one takes the action that maximizes expected utility (MEU) in the current context, sometimes referred to as the *expected utility hypothesis*. Decision theory places this in a dynamic context and says each action should be taken to maximize expected future utility. This is defined recursively, so taken to the limit this implies the optimal future actions need to be determined before the optimal current action can be got via MEU.

Justifications

This section covers basic mathematical justifications of the theory. The best general reference for this is Bernardo and Smith (1994). Additional discussion of prior probabilities appears in ▶ Prior Probability.

Note that Bayesian theory, with its acceptance as a branch of mainstream statistics, is widely accepted for the following reasons:

Application: It has extensive support through practical success, often times by clever combination of prior knowledge and statistical and computational finesse.

Explanation: It provides a convenient common language in which a variety of other theoretical approaches can be represented. For instance PAC, MDL methods, penalized likelihood methods, and the maximum margin approach all find good interpretations within the Bayesian framework.

Composition: It allows different reasoning tasks to be composed in a coherent way. With a probabilistic framework, the components can interoperate in a coherent manner, so that information may flow bidirectionally between components via probabilities.

Composition of processing steps in intelligent systems is a key application for Bayesian methods. For instance, natural language and vision recognition tasks can sometimes be broken down into a processing chain (for instance, doing a named entity recognition step before a dependency parsing step), but these components rarely work conclusively and unambiguously. By attaching probabilities to the output of components, and allowing probabilistic inputs, the uncertainty inherent in individual steps can be propagated and managed.

Theoretical justifications also exist to support each of the different components, probabilities, and utilities. These justifications are based on the concept of *normative* axioms, axioms that do not describe reasoning but rather prescribe basic principles it should follow. The axioms try to capture principles such as coherence and consistency in a quantitative manner. These various justifications have their reported shortcomings and a rich literature exists arguing about the details and postulating new variants. These axiomatic justifications are supportive of the Bayesian approach, but they are not irrefutable.

Justifying probabilities: In the Bayesian approach, beliefs and proportions are given the same mathematical treatment.

One set of arguably controversial justifications for this revolve around betting (Bernardo and Smith 1994, Sect. 2.8.3). Someone's subjective beliefs about specific events, such as significant economic and political events (or horse races), are claimed to be measurable by offering them a series of options or bets. Moreover, if their beliefs do not behave like proportions, then a clever bookmaker can use a so-called Dutch book to consistently profit from them.

An alternative scheme for justifying probability by Cox is based on normative axioms that beliefs should follow. For instance, one controversial axiom by Cox is that belief about a single event should be represented by a single real number. These axioms are presented by Jaynes as rules for a robot (Jaynes 2003), and as rules for intelligent systems by Horvitz et al. (1986).

Justifying decision theory: Another scheme again using normative axioms, by von Neumann and Morgenstern, is used to justify the use of utilities. This scheme assumes probabilities are the basis of inference about uncertainty. A different set of normative axiomatic schemes have been developed that justify the use of probabilities and utilities together under MEU, the best known is by Savage but others exist (Bernardo and Smith 1994).

Bayesian Computation

The first part of this article has been devoted to a brief overview of the Bayesian approach. Computation for Bayesian inference is an extensive field itself. Here we review the basic aspects as a pointer to the literature. This is an active area of research in machine learning, statistics, and a many applied artificial intelligence communities such as natural language processing, image analysis, and others.

In general, in Bayesian reasoning one wants to estimate posterior average parameter values, or their average variance, or some other averaged quantity, then general formulas are given by (in the case of continuous parameters)

$$\bar{\boldsymbol{\Theta}} = \mathcal{E}_{\boldsymbol{\Theta}|D,M,C}[\boldsymbol{\Theta}] = \int_{\boldsymbol{\Theta}} \boldsymbol{\Theta}\, p(\boldsymbol{\Theta}|D,M,C)\mathrm{d}\boldsymbol{\Theta}$$

$$\mathrm{var}(\boldsymbol{\Theta}) = \mathcal{E}_{\boldsymbol{\Theta}|D,M,C}[(\boldsymbol{\Theta} - \bar{\boldsymbol{\Theta}})^2]$$

Marginal likelihood: A useful quantity to assist in evaluating results, and a worthy score in its own right is the marginal likelihood, in the continuous parameter case found from the likelihood $p\,(\boldsymbol{D}|\boldsymbol{\Theta}, M, C)$ by taking an average

$$p(\mathbf{D}|M,C)=\int_{\mathbf{\Theta}} p(\mathbf{\Theta}|M,C)p(\boldsymbol{D}|\mathbf{\Theta},M,C)\mathrm{d}\mathbf{\Theta}$$

This is also called the *normalizing constant* due to its occurrence in the posterior formula

$$p(\mathbf{\Theta}|\boldsymbol{D},M,C) = \frac{p(\mathbf{\Theta}|M,C)p(D|\mathbf{\Theta},M,C)}{p(\mathbf{D}|M,C)}!.$$

It is generally difficult to estimate because of the multidimensional integrals and sums.

Exponential family distributions: Standard probability distributions covered in mathematical statistics, such as the ▶ Gaussian Distribution, the Poisson, Dirichlet, Gamma, and Wishart, have very convenient mathematical properties that make Bayesian estimation easier. With these distributions, one computes statistics, called *sufficient statistics*, such as a mean and sum of squares (for the Gaussian), and then parameter estimation follows with a function inverse on a concave function. This is the basis of ▶ linear regression, ▶ principal components analysis, and some ▶ decision tree learning methods, for instance. All good texts on mathematical statistics cover these in detail. Note the marginal likelihood is often computable in closed form for exponential family distributions.

Graphical models: ▶ Graphical Models are a general family of probabilistic models formed by composing graphs over variables. They work particularly well with exponential family distributions, and allow a rich variety of popular machine learning and data mining methods to be represented and manipulated. Graphical models allow complex models to be composed from simpler components and provide a family of algorithm schemes for developing inference and learning methods that operate on them. They have become the de facto standard for presenting (suitable decomposed) models and algorithms in the machine learning community.

Maximum a posterior estimation: known as MAP, is usually the simplest form of parameter estimation that could be called Bayesian. It also corresponds to a penalized or regularized maximum likelihood method. Given the posterior for a stylized learning problem of the previous section, one finds the parameters $\mathbf{\Theta}$ that maximizes the posterior $p(\mathbf{\Theta},M|\boldsymbol{D},C)$, which can be conveniently done without computing the marginal likelihood above, so

$$\widehat{\mathbf{\Theta}_{MP}} = \underset{\mathbf{\Theta}}{\text{armugmax}}\ \log\ p(\mathbf{\Theta},\boldsymbol{D}|M,C)$$

where the log probability can be broken down as a prior and a likelihood term

$$\log p(\mathbf{\Theta},\boldsymbol{D}|M,C) = \log p(\mathbf{\Theta}|M,C) + \log p(\boldsymbol{D}|\mathbf{\Theta},M,C).$$

The Laplace approximation: When the posterior is well behaved, and there is a large amount of data, the posterior is focused around a vanishing small region in parameter space of diameter $O(1/\sqrt{(N)})$. If this occurs away from the boundary of the parameter space, then one can make a second-order Taylor expansion of the log. posterior at the MAP point and the result is a Gaussian approximation to the posterior.

$$\log p(\boldsymbol{D},\mathbf{\Theta}|M,C) \approx \log p(\boldsymbol{D},\widehat{\mathbf{\Theta}_{MP}}|M,C) + \frac{1}{2}(\widehat{\mathbf{\Theta}_{MP}}-\mathbf{\Theta})^T \left.\frac{\mathrm{d}^2\log p(\boldsymbol{D},\mathbf{\Theta}|M,C)}{\mathrm{d}\mathbf{\Theta}\mathrm{d}\mathbf{\Theta}^T}\right|_{\mathbf{\Theta}=\widehat{\mathbf{\Theta}_{MP}}} (\widehat{\mathbf{\Theta}_{MP}}-\mathbf{\Theta}).$$

From this, one can approximate integrals such as the marginal likelihood $p(\boldsymbol{D}|M,C)$. This is known as the *Laplace approximation*, the name of the corresponding general method used for the asymptotic expansion of integrals. In general, this is a poor approximation, but it serves to aid our understanding of parameter estimation (MacKay 2003, Chaps. 27 and 28), and is the approximate basis for some model selection criteria.

Latent variable models: Latent variables are data that are hidden and thus never observed in the evidence. However, their existence is postulated as a significant component of the model. For instance, in ▶ Clustering (an unsupervised method) and finite mixture models generally, one assumes each data point has a hidden class label, thus the Bayesian model of clustering is a simple kind of latent variable model.

▶ *Markov chain Monte Carlo* methods: The most general form of reasoning and estimation available are the *Markov chain Monte Carlo* (MCMC) methods. The MCMC methods couple two processes: first, they use *Monte Carlo* or simulation methods to estimate the integral, and second they use a *Markov Chain* to sample, so sampling is sequentially (Markovian) based, and samples are not independent.

Simulation methods generally use the functional form of $p(\boldsymbol{\Theta}, \boldsymbol{D}|M, C)$ so we do not need to compute the marginal likelihood. Hence, given a set of I samples $\{\boldsymbol{\Theta}_1, \ldots, \boldsymbol{\Theta}_I\}$ the expected value is approximated with a weighted average

$$\bar{\boldsymbol{\Theta}} \approx \frac{1}{I}\sum_{i=1}^{I} w_i \boldsymbol{\Theta}_i .$$

The simplest case is where the samples are made independently according to the posterior itself and then the weights $w_i = 1$, This is called the ordinary Monte Carlo (OMC) method, but it is not often usable in practice because efficient multidimensional posterior samplers rarely exist. Alternatively, one can sample according to a Markov Chain, $\boldsymbol{\Theta}_{i+1} \sim q(\boldsymbol{\Theta}_{i+1}|\boldsymbol{\Theta}_i)$, so each $\boldsymbol{\Theta}_{i+1}$ is conditionally dependent on $\boldsymbol{\Theta}i$. So while samples are not independent, as long as the long run distribution of the Markov chain is the same as the posterior, the same approximation formula holds. There are a rich variety of MCMC methods, and this forms one of the key areas of current research.

Gibbs sampling: The simplest kind of MCMC method samples each dimension (or sub-vector) in turn. Suppose the parameter vector has K real components, $\boldsymbol{\Theta} = (\theta_1, \ldots, \theta_K)$. Sampling a complete $\boldsymbol{\Theta}$ in one go is not generally possible given just a functional form of the posterior $p(\boldsymbol{\Theta}|\boldsymbol{D}, M, C)$ but given no computable form for the normalizing constant. Gibbs sampling works in the one-dimensional case where normalizing bounds can be obtained and sampling tricks used. The conditional posterior of θ_k is given by

$$p(\theta_k|(\theta_1, \ldots, \theta_{k-1}, \theta_{k+1}, \ldots, \theta k), \boldsymbol{D}, M, C),$$

and this is usually easier to sample from.

The Gibbs (and MCMC) sample $\boldsymbol{\Theta}_{i+1}$ can be drawn given the previous sample $\boldsymbol{\Theta}_i$ by progressively resampling each dimension in turn and so slowly updating the full vector:

1. Sample $\theta_{i+1,1}$ according to $p(\theta_1|\theta_{i,2}, \ldots, \theta_{i,K}, \mathbf{D}, M, C)$. ...

k. Sample $\theta_{i+1,k}$ according to $p(\theta_2|\theta i_{+1,1}, \ldots, \theta_{i+1,k-1}, \theta_{i,k+1}, \ldots, \theta_{i,K}, \mathbf{D}, M, C)$.

...

k. Sample $\theta_{i+1,k}$ according to $p(\theta_K|\theta_{i+1,1}, \ldots, \theta_{i+1,K-1}, \mathbf{D}, M, C)$.

In samping terms, this method is no more successful than coordinate-wise ascent is as a primitive greedy search method: it is supported by theoretical results but can be very slow to converge.

Variational approximations: When the function you seek to optimize or average over presents difficulty, perhaps it is highly multimodal, then one option is to change the function itself, and replace it with a more readily approximated function. Variational methods provide a general principle for doing this safely. The general principle uses variational calculus, which is the calculus over functions, not just variables. Variational methods are a very general approach that can be used to develop a broad range of algorithms (Wainwright and Jordan 2008).

Nonparametric models: The above discussion implicitly assumed the model has a fixed finite parameter vector $\boldsymbol{\Theta}$. If one is attempting to model a regression function, or a language grammar, or image model of unknown a priori structural complexity, then one cannot know the dimension ahead of time. Moreover, as in the case of functions, the dimension cannot always be finite. The ▶ Bayesian Nonparametric Models address this situation, and are perhaps the most important family of techniques for general machine learning.

Cross-References

▶ Bayes' Rule
▶ Bayesian Nonparametric Models

▶ Markov Chain Monte Carlo
▶ Prior Probability

Recommended Reading

A good introduction to the problems of uncertainty and philosophical issues behind the Bayesian treatment of probability is in Lindley (2006). From the statistical machine learning perspective, a good introductory text is by MacKay (2003) who carefully covers information theory, probability, and inference but not so much statistical machine learning. Another alternative introduction to probabilities is the posthumously completed and published work of Jaynes (2003).

Discussions from the frequentist versus Bayesian battlefront can be found in works such as Rosenkrantz (1983), and in the approximate artificial intelligence versus probabilistic battlefront in discussion articles such as Cheeseman's (1988) and the many responses and rebuttals. It should be noted that it is the continued success in applications that have really led these methods into the mainstream, not the entertaining polemics.

Good mathematical statistics text books, such as Casella and Berger (2001) cover the breadth of statistical methods and therefore handle basic Bayesian theory. A more comprehensive treatment is given in Bayesian texts such as Gelman et al. (2003).

Most advanced statistical machine learning text books cover Bayesian methods, but to fully understand the subtleties of prior beliefs and Bayesian methodology one needs to view more advanced Bayesian literature. A detailed theoretical reference for Bayesian methods is Bernardo and Smith (1994).

Bernardo J, Smith A (1994) Bayesian theory. Wiley, Chichester
Casella G, Berger R (2001) Statistical inference, 2nd edn. Duxbury, Pacific Grove
Cheeseman P (1988) An inquiry into computer understanding. Comput Intell 4(1):58–66
Gelman A, Carlin J, Stern H, Rubin D (2003) Bayesian data analysis, 2nd edn. Chapman & Hall/CRC Press, Boca Raton
Horvitz E, Heckerman D, Langlotz C (1986) A framework for comparing alternative formalisms for plausible reasoning. In: Fifth national conference on artificial intelligence, Philadelphia, pp 210–214
Jaynes E (2003) Probability theory: the logic of science. Cambridge University Press, New York
Lindley D (2006) Understanding uncertainty. Wiley, Hoboken
MacKay D (2003) Information theory, inference, and learning algorithms. Cambridge University Press, Cambridge
Rosenkrantz R (ed) (1983) E.T. Jaynes: papers on probability, statistics and statistical physics. D. Reidel, Dordrecht
Wainwright MJ, Jordan MI (2008) Graphical models, exponential families, and variational inference. Now Publishers, Hanover

Bayesian Model Averaging

▶ Learning Graphical Models

Bayesian Network

Synonyms

Bayes net

Definition

A Bayesian network is a form of directed ▶ graphical model for representing multivariate probability distributions.

The nodes of the network represent a set of random variables, and the directed arcs represent causal relationships between variables. The *Markov property* is usually required: every direct dependency between a possible cause and a possible effect has to be shown with an arc. Bayesian networks with the Markov property are called *I-maps* (independence maps). If all arcs in the network correspond to a direct dependence on the system being modeled, then the network is said to be a *D-map* (dependence-map). Each node is associated with a conditional probability distribution, that quantifies the effects the parents of the node, if any, have on it. Bayesian support various forms of reasoning: *diagnosis*, to derive causes from symptoms, *prediction*, to derive effects from causes, and *intercausal reasoning*, to discover the mutual causes of a common effect.

Cross-References

▸ Graphical Models

Bayesian Nonparametric Models

Peter Orbanz[1] and Yee Whye Teh[2]
[1]Cambridge University, Cambridge, UK
[2]University College London, London, UK

Synonyms

Bayesian methods; Dirichlet process; Gaussian processes; Prior probabilities

Definition

A Bayesian nonparametric model is a Bayesian model on an infinite-dimensional parameter space. The parameter space is typically chosen as the set of all possible solutions for a given learning problem. For example, in a regression problem, the parameter space can be the set of continuous functions, and in a density estimation problem, the space can consist of all densities. A Bayesian nonparametric model uses only a finite subset of the available parameter dimensions to explain a finite sample of observations, with the set of dimensions chosen depending on the sample such that the effective complexity of the model (as measured by the number of dimensions used) adapts to the data. Classical adaptive problems, such as nonparametric estimation and model selection, can thus be formulated as Bayesian inference problems. Popular examples of Bayesian nonparametric models include Gaussian process regression, in which the correlation structure is refined with growing sample size, and Dirichlet process mixture models for clustering, which adapt the number of clusters to the complexity of the data. Bayesian nonparametric models have recently been applied to a variety of machine learning problems, including regression, classification, clustering, latent variable modeling, sequential modeling, image segmentation, source separation, and grammar induction.

Motivation and Background

Most of machine learning is concerned with learning an appropriate set of parameters within a model class from ▸ training data. The meta-level problems of determining appropriate model classes are referred to as model selection or model adaptation. These constitute important concerns for machine learning practitioners, not only for avoidance of over-fitting and under-fitting, but also for discovery of the causes and structures underlying data. Examples of model selection and adaptation include selecting the number of clusters in a clustering problem, the number of hidden states in a hidden Markov model, the number of latent variables in a latent variable model, or the complexity of features used in nonlinear regression.

Nonparametric models constitute an approach to model selection and adaptation where the sizes of models are allowed to grow with data size. This is as opposed to *parametric models*, which use a fixed number of parameters. For example, a parametric approach to density estimation would be to fit a Gaussian or a mixture of a fixed number of Gaussians by maximum likelihood. A nonparametric approach would be a Parzen window estimator, which centers a Gaussian at each observation (and hence uses one mean parameter per observation). Another example is the support vector machine with a Gaussian kernel. The representer theorem shows that the decision function is a linear combination of Gaussian radial basis functions centered at every input vector, and thus has a complexity that grows with more observations. Nonparametric methods have long been popular in classical (non-Bayesian) statistics (Wasserman 2006). They often perform impressively in applications and, though theoretical results for such models are typically harder to prove than for parametric models, appealing theoretical properties have been established for a wide range of models.

Bayesian nonparametric methods provide a Bayesian framework for model selection and adaptation using nonparametric models. A Bayesian formulation of nonparametric problems is nontrivial, since a Bayesian model defines prior and posterior distributions on a single fixed parameter space, but the dimension of the parameter space in a nonparametric approach should change with sample size. The Bayesian nonparametric solution to this problem is to use an infinite-dimensional parameter space, and to invoke only a finite subset of the available parameters on any given finite data set. This subset generally grows with the data set. In the context of Bayesian nonparametric models, "infinite-dimensional" can therefore be interpreted as "of finite but unbounded dimension." More precisely, a Bayesian nonparametric model is a model that (1) constitutes a Bayesian model on an infinite-dimensional parameter space and (2) can be evaluated on a finite sample in a manner that uses only a finite subset of the available parameters to explain the sample.

We make the above description more concrete in the next section when we describe a number of standard machine learning problems and the corresponding Bayesian nonparametric solutions. As we will see, the parameter space in (1) typically consists of functions or of measures, while (2) is usually achieved by marginalizing out surplus dimensions over the prior. Random functions and measures and, more generally, probability distributions on infinite-dimensional random objects are called *stochastic processes*; examples that we will encounter include Gaussian processes, Dirichlet processes, and beta processes. Bayesian nonparametric models are often named after the stochastic processes they contain. The examples are then followed by theoretical considerations, including formal constructions and representations of the stochastic processes used in Bayesian nonparametric models, exchangeability, and issues of consistency and convergence rate. We conclude this chapter with future directions and a list of literature available for reading.

Examples

Clustering with mixture models. Bayesian nonparametric generalizations of finite mixture models provide an approach for estimating both the number of components in a mixture model and the parameters of the individual mixture components simultaneously from data. Finite mixture models define a density function over data items x of the form $p(x) = \sum_{k=1}^{K} \pi_k p(x|\theta_k)$, where π_k is the mixing proportion and θ_k are parameters associated with component k. The density can be written in a non-standard manner as an integral: $p(x) = \int p(x|\theta)G(\theta)d\theta$, where $G = \sum_{k=1}^{K} \pi_k \delta_{\theta_k}$ is a discrete mixing distribution encapsulating all the parameters of the mixture model and δ_θ is a dirac distribution (atom) centered at θ. Bayesian nonparametric mixtures use mixing distributions consisting of a *countably infinite* number of atoms instead:

$$G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k}. \quad (1)$$

This gives rise to mixture models with an infinite number of components. When applied to a finite training set, only a finite (but varying) number of components will be used to model the data, since each data item is associated with exactly one component but each component can be associated with multiple data items. Inference in the model then automatically recovers both the number of components to use and the parameters of the components. Being Bayesian, we need a prior over the mixing distribution G, and the most common prior to use is a *Dirichlet process* (DP). The resulting mixture model is called a DP mixture.

Formally, a Dirichlet process $DP(\alpha, H)$ parametrized by a concentration paramter $\alpha > 0$ and a base distribution H is a prior over distributions (probability measures) G such that, for any finite partition $A_1, \ldots, A_m$ of the parameter space, the induced random vector $(G(A_1), \ldots, G(A_m))$ is Dirichlet distributed with parameters $(\alpha H(A_1), \ldots, \alpha H(A_m))$ (see entitled Section "Theory" for a discussion of subtleties involved in this definition). It can be shown that draws from a DP will be discrete distributions as given in (1). The DP also induces a distribution over partitions of integers called the *Chinese restaurant process* (CRP), which directly describes the prior over how data items are clustered under the DP mixture. For more details on the DP and the CRP, see ▸ Dirichlet Process.

Nonlinear regression. The aim of regression is to infer a continuous function from a training set consisting of input–output pairs $\{(t_i, x_i)\}i_{=1}^{n}$. Parametric approaches parametrize the function using a finite number of parameters and attempt to infer these parameters from data. The prototypical Bayesian nonparametric approach to this problem is to define a prior distribution over continuous functions directly by means of a *Gaussian process* (GP). As explained in the Chapter ▸ Gaussian Process, a GP is a distribution on an infinite collection of random variables X_t, such that the joint distribution of each finite subset $X_{t_1}, \ldots, X_{t_m}$ is a multivariate Gaussian. A value x_t taken by the variable X_t can be regarded as the value of a continuous function f at t, that is, $f(t) = x_t$. Given the training set, the Gaussian process posterior is again a distribution on functions, conditional on these functions taking values $f(t_1) = x_1, \ldots, f(t_n) = x_n$.

Latent feature models. These models represent a set of objects in terms of a set of latent features, each of which represents an independent degree of variation exhibited by the data. Such a representation of data is sometimes referred to as a distributed representation. In analogy to nonparametric mixture models with an unknown number of clusters, a Bayesian nonparametric approach to latent feature modeling allows for an unknown number of latent features. The stochastic processes involved here are known as the *Indian buffet process* (IBP) and the *beta process* (BP). Draws from BPs are random discrete measures, where each of an infinite number of atoms has a mass in (0, 1) but the masses of atoms need not sum to 1. Each atom corresponds to a feature, with the mass corresponding to the probability that the feature is present for an object. We can visualize the occurrences of features among objects using a binary matrix, where the (i, k) entry is 1 if object i has feature k and 0 otherwise. The distribution over binary matrices induced by the BP is called the IBP.

▸ *Hidden Markov Models* (*HMMs*). HMMs are popular models for sequential or temporal data, where each time step is associated with a state, with state transitions dependent on the previous state. An *infinite HMM* is a Bayesian nonparametric approach to HMMs, where the number of states is unbounded and allowed to grow with the sequence length. It is defined using one DP prior for the transition probabilities going out from each state. To ensure that the set of states reachable from each outgoing state is the same, the base distributions of the DPs are shared and given a DP prior recursively. The construction is called a *hierarchical Dirichlet process* (HDP); see below.

▸ *Density Estimation*. A nonparametric Bayesian approach to density estimation requires a prior on densities or distributions. However, the DP is not useful in this context, since it generates discrete distributions. A useful density estimator should smooth the empirical density (such as a Parzen window estimator), which requires a prior that can generate smooth distributions. Priors applicable in density estimation problems include DP mixture models and Pólya trees.

If $p(x|\theta)$ is a smooth density function, the density $\sum_{k=1}^{\infty} \pi_k p(x|\theta_k)$ induced by a DP mixture model is a smooth random density, such that DP mixtures can be used as prior in density estimation problems.

Pólya trees are priors on probability distributions that can generate both discrete and piecewise continuous distributions, depending on the choice of parameters. Pólya trees are defined by a recursive infinitely deep binary subdivision of the domain of the generated random measure. Each subdivision is associated with a beta random variable which describes the relative amount of mass on each side of the subdivision. The DP is a special case of a Pólya tree corresponding to a particular parametrization. For other parametrizations the resulting random distribution can be smooth, so it is suitable for density estimation.

Power-law Phenomena. Many naturally occurring phenomena exhibit power-law behavior. Examples include natural languages, images, and social and genetic networks. An interesting generalization of the DP, called the *Pitman-Yor process*, PYP(α, d, H), has recently been successfully used to model power-law data. The Pitman-Yor process augments the DP by a third parameter $d \in [0, 1)$. When $d = 0$ the PYP is a $DP(\alpha, H)$, while when $\alpha = 0$ it is a so called *normalized stable process*.

Sequential modeling. HMMs model sequential data using latent variables representing the underlying state of the system, and assuming that each state only depends on the previous state (the so called Markov property). In some applications, for example language modeling and text compression, we are interested in directly modeling sequences without using latent variables, and without making any Markov assumptions, i.e., modeling each observation conditional on all previous observations in the sequence. Since the set of potential sequences of previous observations is unbounded, this calls for nonparametric models. A *hierarchical Pitman-Yor process* can be used to construct a Bayesian nonparametric solution whereby the conditional probabilities are coupled hierarchically.

Dependent and hierarchical models. Most of the Bayesian nonparametric models described so far are applied in settings where observations are homogeneous or exchangeable. In many real world settings observations are not homogeneous, and in fact are often structured in interesting ways. For example, the data generating process might change over time thus observations at different times are not exchangeable, or observations might come in distinct groups with those in the same group being more similar than across groups.

Significant recent efforts in Bayesian nonparametrics research have been placed in developing extensions that can handle these non-homogeneous settings. Dependent Dirichlet processes are stochastic processes, typically over a spatial or temporal domain, which define a Dirichlet process (or a related random measure) at each point with neighboring DPs being more dependent. These are used for spatial modeling, nonparametric regression, as well as for modeling temporal changes. Alternatively, hierarchical Bayesian nonparametric models like the hierarchical DP aim to couple multiple Bayesian nonparametric models within a hierarchical Bayesian framework. The idea is to allow sharing of statistical strength across multiple groups of observations. Among other applications, these have been used in the infinite HMM, topic modeling, language modeling, word segmentation, image segmentation, and grammar induction. For an overview of various dependent Bayesian nonparametric models and their applications in biostatistics please refer to Dunson (2010). Teh and Jordan (2010) is an overview of hierarchical Bayesian nonparametric models as well as a variety of applications in machine learning.

Theory

As we saw in the preceding examples, Bayesian nonparametric models often make use of priors over functions and measures. Because these spaces typically have uncountable number of dimensions, extra care has to be taken to define the priors properly and to study the asymptotic properties of estimation in the resulting models. In this section we give an overview of the basic concepts involved in the theory of Bayesian nonparametric models. We start with a discussion of the importance of exchangeability in Bayesian parametric and nonparametric statistics. This is followed by representations of the priors and issues of convergence.

Exchangeability

The underlying assumption of all Bayesian methods is that the parameter specifying the observation model is a random variable. This assumption is subject to much criticism, and at the heart of the Bayesian versus non-Bayesian debate that has long divided the statistics community. However, there is a very general type of observation for which the existence of such a random variable can be derived mathematically: For so-called *exchangeable* observations, the Bayesian assumption that a randomly distributed parameter exists is not a modeling assumption, but a mathematical consequence of the data's properties.

Formally, a sequence of variables X_1, $X_2, \ldots, X_n$ over the same probability space (χ, Ω) is *exchangeable* if their joint distribution is invariant to permuting the variables. That is, if P is the joint distribution and σ any permutation of $\{1, \ldots, n\}$, then

$$P(X_1 = x_1, X_2 = x_2 \ldots X_n = x_n) \\ = P(X_1 = x_{\sigma(1)}, X_2 = x_{\sigma(2)} \ldots X_n = x_{\sigma(n)}). \tag{2}$$

An infinite sequence X_1, X_2, ...is *infinitely exchangeable* if X_1, ..., X_n is exchangeable for *everyn* $\geqslant$ 1. In this chapter, we mean infinite exchangeability whenever we write exchangeability. Exchangeability reflects the assumption that the variables do not depend on their indices although they may be dependent among themselves. This is typically a reasonable assumption in machine learning and statistical applications, even if the variables are not themselves independently and identically distributed (iid).

Exchangeability is a much weaker assumption than iid since iid variables are automatically exchangeable.

If θ parametrizes the underlying distribution, and one assumes a prior distribution over θ, then the resulting marginal distribution over X_1, X_2,...with θ marginalized out will still be exchangeable. A fundamental result credited to de Finetti (1931) states that the converse is also true. That is, if X_1, X_2, ...is (infinitely) exchangeable, then there is a random θ such that:

$$P(X_1, \ldots, X_n) = \int P(\theta) \prod_{i=1}^{n} P(X_i|\theta) d\theta \tag{3}$$

for every $n \geqslant 1$. In other words, the seemingly innocuous assumption of exchangeability automatically implies the existence of a hierarchical Bayesian model with θ being the random latent parameter. This the crux of the fundamental importance of exchangeability to Bayesian statistics.

In de Finetti's Theorem it is important to stress that θ can be infinite dimensional (it is typically a random measure), thus the hierarchical Bayesian model (3) is typically a nonparametric one. For an example, the Blackwell–MacQueen urn scheme (related to the CRP) is exchangeable and thus implicitly defines a random measure, namely the DP (see ▸ Dirichlet Processes for more details). In this sense, we will see below that de Finetti's theorem is an alternative route to Kolmogorov's extension theorem, which implicitly defines the stochastic processes underlying Bayesian nonparametric models.

Model Representations

In finite dimensions, a probability model is usually defined by a density function or probability mass function. In infinite dimensional spaces, this approach is not generally feasible, for reasons explained below. To define or work with a Bayesian nonparametric model, we have to choose alternative mathematical representations.

Weak distributions. A weak distribution is a representation for the distribution of a stochastic process, that is, for a probability distribution on an infinite-dimensional sample space. If we assume that the dimensions of the space are indexed by $t \in T$, the stochastic process can be regarded as the joint distribution P of an infinite set of random variables $\{Xt\}_{t \in T}$. For any finite subset $S \subset T$ of dimensions, the joint distribution P_S of the corresponding subset $\{X_t\}_{t \in S}$ of random variables is a finite-dimensional marginal of P. The *weak distribution* of a stochastic process is the set of all its finite-dimensional marginals, that is, the set $\{P_S : S \subset T, |S| < \infty\}$. For example, the customary definition of the Gaussian process as an infinite collection of random variables, each finite subset of which has a joint Gaussian distribution, is an example of a weak distribution representation. In contrast to the explicit representations to be described below, this representation is generally not generative, because it represents the distribution rather than a random draw, but is more widely applicable.

Apparently, just defining a weak distribution in this manner need not imply that it is a valid representation of a stochastic process. A given collection of finite-dimensional distributions represents a stochastic process only (1) if a process with these distributions as its marginals actually exists, and (2) if it is uniquely defined by the marginals. The mathematical result which guarantees that weak distribution representations are valid is the *Kolmogorov extension theorem* (also known as the Daniell–Kolmogorov theorem or the Kolmogorov consistency theorem). Suppose that a collection $\{P_S : S \subset T, |S| < \infty\}$ of distributions is given. If all distributions in the collection are marginals of each other, that is, if P_{s_1} is a marginal of P_{s_2} whenever $S_1 \subset S_2$, the set of distributions is called a *projective family*. The Kolmogorov extension theorem states that, if the set T is countable, and if the distributions P_S form a projective family, then there exists a uniquely defined stochastic process with the collection $\{P_S\}$ as its marginal distributions. In other words, any projective family for a countable set T of dimensions is the weak distribution of a stochastic process. Conversely, any stochastic process can be represented in this manner, by computing its set of finite-dimensional marginals.

The weak distribution representation assumes that all individual random variable X_t of the stochastic process take values in the same sample space Ω. The stochastic process P defined by the weak distribution is then a probability distribution on the sample space Ω^T, which can be interpreted as the set of all functions $f : T \to \Omega$. For example, to construct a GP we might choose $T = \mathbb{Q}$ and $\Omega = \mathbb{R}$ to obtain real-valued functions on the countable space of rational numbers. Since $\mathbb{Q}$ is dense in $\mathbb{R}$, the function f can then be extended to all of $\mathbb{R}$ by continuity. To define the DP as a distribution over probability measures on $\mathbb{R}$, we note that a probability measure is a set function that maps "random events," i.e., elements of the Borel σ-algebra $\mathcal{B}(\mathbb{R})$ of $\mathbb{R}$, into probabilities in [0, 1]. We could therefore choose a weak distribution consisting of Dirichlet distributions, and set $T = \mathcal{B}(\mathbb{R})$ and $\Omega = [0, 1]$. However, this approach raises a new problem because the set $\mathcal{B}(\mathbb{R})$ is not countable. As in the GP, we can first define the DP on a countable "base" for $\mathcal{B}(\mathbb{R})$ then extend to all random events by continuity of measures. More precise descriptions are unfortunately beyond the scope of this chapter.

Explicit representations. Explicit representations directly describe a random draw from a stochastic process, rather than its distribution. A prominent example of an explicit representation is the so-called *stick-breaking representation* of the Dirichlet process. The discrete random measure G in (1) is completely determined by the two infinite sequences $\{\pi_k\}_{k\in\mathbb{N}}$ and $\{\theta_k\}_{k\in\mathbb{N}}$. The stick-breaking representation of the DP generates these two sequences by drawing $\theta k \sim H$ iid and $v_k \sim Beta(1, \alpha)$ for $k = 1, 2, \ldots$. The coefficients πk are then computed as $\pi_k = v_k \prod_{j=1}^{k-1}(1 - v_k)$. The measure G so obtained can be shown to be distributed according to a $DP(\alpha, G_0)$. Similar representations can be derived for the Pitman–Yor process and the beta process as well. Explicit representations, if they exist for a given model, are typically of great practical importance for the derivation of algorithms.

Implicit Representations. A third representation of infinite dimensional models is based on de Finetti's Theorem. Any exchangeable sequence $X_1, \ldots, X_n$ uniquely defines a stochastic process θ, called the de Finetti measure, making the X_i's iid. If the X_i's are sufficient to define the rest of the model and their conditional distributions are easily specified, then it is sufficient to work directly with the X_i's and have the underlying stochastic process implicitly defined. Examples include the Chinese restaurant process (an exchangeable distribution over partitions) with the DP as the de Finetti measure, and the Indian buffet process (an exchangeable distribution over binary matrices) with the BP being the corresponding de Finetti measure. These implicit representations are useful in practice as they can lead to simple and efficient inference algorithms.

Finite representations. A fourth representation of Bayesian nonparametric models is as the infinite limit of finite (parametric) Bayesian models. For example, DP mixtures can be derived as the infinite limit of finite mixture models with particular Dirichlet priors on mixing proportions, GPs can be derived as the infinite limit of particular Bayesian regression models with Gaussian priors, while BPs can be derived as from the limit of an infinite number of independent beta variables. These representations are sometimes more intuitive for practitioners familiar with parametric models. However, not all Bayesian nonparametric models can be expressed in this fashion, and they do not necessarily make clear the mathematical subtleties involved.

Consistency and Convergence Rates

A recent series of works in mathematical statistics examines the convergence properties of Bayesian nonparametric models, and in particular the questions of *consistency* and *convergence rates*. In this context, a Bayesian model is called consistent if, given that an infinite amount of data is available, the model posterior will concentrate in a neighborhood of the true solution (e.g., true function or density). A rate of convergence specifies, for a finite sample, how rapidly the posterior concentrates depending on the sample size. In their pioneering article Diaconis and Freedman (1986) showed, to the great surprise of much of the Bayesian community, that models such as the Dirichlet process can be inconsistent, and may converge to arbitrary solutions even for an infinite amount of data.

More recent results, notably by van der Vaart and Ghosal, apply modern methods of mathematical statistics to study the convergence properties of Bayesian nonparametric models (see e.g., Gho (2010) and references therein). Consistency has been shown for a number of models, including Gaussian processes and Dirichlet process mixtures. However, a particularly interesting aspect of this line of work are results on convergence rates, which specify the rate of concentration of the posterior depending on sample size, on the complexity of the model, and on how much probability mass the prior places around the true solution. To make such results quantitative requires a measure for the complexity of a Bayesian nonparametric model. This is done by means of complexity measures developed in empirical process theory and statistical learning theory, such as metric entropies, covering numbers and bracketing, some of which are well-known in theoretical machine learning.

Inference

There are two aspects to inference from Bayesian nonparametric models: the analytic tractability of posteriors for the stochastic processes embedded in Bayesian nonparametric models, and practical inference algorithms for the overall models. Bayesian nonparametric models typically include stochastic processes such as the Gaussian process and the Dirichlet process. These processes have an infinite number of dimensions, hence naïve algorithmic approaches to computing posteriors are generally infeasible. Fortunately, these processes typically have analytically tractable posteriors, so all but finitely many of the dimensions can be analytically integrated out efficiently. The remaining dimensions, along with the parametric parts of the models, can then be handled by the usual inference techniques employed in parametric Bayesian modeling, including Markov chain Monte Carlo, sequential Monte Carlo, variational inference, and message-passing algorithms like expectation propagation. The precise choice of approximations to use will depend on the specific models under consideration, with speed/accuracy trade-offs between different techniques generally following those for parametric models. In the following, we will give two examples to illustrate the above points, and discuss a few theoretical issues associated with the analytic tractability of stochastic processes.

Examples

In Gaussian process regression, we model the relationship between an input x and an output y using a function f, so that $y \sim f(x) + \varepsilon$, where ε is iid Gaussian noise. Given a GP prior over f and a finite training data set $\{(x_i, y_i)\}_{i=1}^{n}$ we wish to compute the posterior over f. Here we can use the weak representation of f and note that $\{f(xi)\}_{i=1}^{n}$ is simply a finite-dimensional Gaussian with mean and covariance given by the mean and covariance functions of the GP. Inference for $\{f(x_i)\}_{i=1}^{n}$ is then straightforward. The approach can be thought of equivalently as marginalizing out the whole function except its values on the training inputs. Note that although we only have the posterior over $\{f(x_i)\}_{i=1}^{n}$, this is sufficient to reconstruct the function evaluated at any other point x_0 (say the test input), since $f(x_0)$ is Gaussian and independent of the training data $\{(x_i, y_i)\}_{i=1}^{n}$ given $\{f(x_i)\}_{i=1}^{n}$. In GP regression the posterior over $\{f(x_i)\}_{i=1}^{n}$ can be computed exactly. In GP classification or other regression settings with nonlinear likelihood functions, the typical approach is to use sparse methods based on variational approximations or expectation propagation; see Chapter ▶ Gaussian Process for details.

Our second example involves Dirichlet process mixture models. Recall that the DP induces a clustering structure on the data items. If our training set consists of n data items, since each item can only belong to one cluster, there are at most n clusters represented in the training set. Even though the DP mixture itself has an infinite number of potential clusters, all but finitely many of these are not associated with data, thus the associated variables need not be explicitly represented at all. This can be understood either as marginalizing out these variables, or as an implicit representation which can be made explicit whenever required by sampling from the prior. This idea is applicable for DP mixtures using both the Chinese restaurant process and the stick-breaking representations. In the CRP representation, each data item x_i is associated with a cluster index z_i, and each cluster k with a parameter θ_k^* (these parameters can be marginalized out if H is conjugate to F), and these are the only latent variables that need be represented in memory. In the stick-breaking representation, clusters are ordered by decreasing prior expected size, with cluster k associated with a parameter θ_k^* and a size π_k. Each data item is again associated with a cluster index z_i, and only the clusters up to $K = \max(z_1, \ldots, zn)$ need to be represented. All clusters with index $> K$ need not be represented since their posterior conditioning on $\{(x_i, z_i)\}_{i=1}^{n}$ is just the prior.

On Bayes Equations and Conjugacy

It is worth noting that the posterior of a Bayesian model is, in abstract terms, defined as the conditional distribution of the parameter given the data and the hyperparameters, and this definition does not require the existence of a Bayes equation. If a Bayes equation exists for the model, the posterior can equivalently be defined as the left-hand side of the Bayes equation. However, for some stochastic processes, notably the DP on an uncountable space such as $\mathbb{R}$, it is not possible to define a Bayes equation even though the posterior is still a well-defined mathematical object. Technically speaking, existence of a Bayes equation requires the family of all possible posteriors to be dominated by the prior, but this is not the case for the DP. That posteriors of these stochastic processes can be evaluated at all is solely due to the fact that they admit an analytic representation.

The particular form of tractability exhibited by many stochastic processes in the literature is that of a *conjugate* posterior, that is, the posterior belongs to the same model family as the prior, and the posterior parameters can be computed as a function of the prior hyperparameters and the observed data. For example, the posterior of a $DP(\alpha, G_0)$ under observations $\theta_1, \ldots, \theta n$ is again a Dirichlet process, $DP(\alpha + n, \frac{1}{\alpha+n}(\alpha G_0 + \sum \delta_{\theta_i}))$. Similarly the posterior of a GP under observations of $f(x_1), \ldots, f(x_n)$ is still a GP. It is this conjugacy that allows practical inference in the examples above. A Bayesian nonparametric model is conjugate if and only if the elements of its weak distribution, i.e., its finite-dimensional marginals, have a conjugate structure as well (Orbanz 2010). In particular, this characterizes a class of conjugate Bayesian nonparametric models whose weak distributions consist of exponential family models. Note however, that lack of conjugacy does not imply intractable posteriors. An example is given by the Pitman–Yor process in which the posterior is given by a sum of a finite number of atoms and a Pitman-Yor process independent from the atoms.

Future Directions

Since MCMC (see ▸ Markov Chain Monte Carlo) sampling algorithms for Dirichlet process mixtures became available in the 1990s and made latent variable models with nonparametric Bayesian components applicable to practical problems, the development of Bayesian nonparametrics has experienced explosive growth (Escobar and West 1995; Neal 2000). Arguably, though, the results available so far have only scratched the surface. The repertoire of available models is still mostly limited to using the Gaussian process, the Dirichlet process, the beta process, and generalizations derived from those. In principle, Bayesian nonparametric models may be defined on any infinite-dimensional mathematical object of possible interest to machine learning and statistics. Possible examples are kernels, infinite graphs, special classes of functions (e.g., piece-wise continuous or Sobolev functions), and permutations.

Aside from the obvious modeling questions, two major future directions are to make Bayesian nonparametric methods available to a larger audience of researchers and practitioners through the development of software packages, and to understand and quantify the theoretical properties of available methods.

General-Purpose Software Package

There is currently significant growth in the application of Bayesian nonparametric models across a variety of application domains both in machine learning and in statistics. However significant hurdles still exist, especially the expense and expertise needed to develop computer programs for inference in these complex models. One future direction is thus the development of software packages that can compile efficient inference algorithms automatically given model specifications, thus allowing a much wider range of modeler to make use of these models. Current developments include the R DPpackage (http://cran.r-project.org/web/packages/DPpackage), the hierarchical Bayesian compiler (http://www.cs.utah.edu/hal/HBC), adaptor grammars (http://www.cog.brown.edu/mj/Software.htm), the MIT-Church project (http://projects.csail.mit.edu/church/wiki/Church), as well as efforts to add Bayesian nonparametric models to the repertoire of current Bayesian modeling environments like OpenBugs (http://mathstat.helsinki.fi/openbugs) and infer.NET (http://research.microsoft.com/en-us/um/cambridge/projects/infernet).

Statistical Properties of Models

Recent work in mathematical statistics provides some insight into the quantitative behavior of Bayesian nonparametric models (cf theory section). The elegant, methodical approach underlying these results, which quantifies model complexity by means of empirical process theory and then derives convergence rates as a function of the complexity, should be applicable to a wide range of models. So far, however, only results for Gaussian processes and Dirichlet process mixtures have been proven, and it will be of great interest to establish properties for other priors. Some models developed in machine learning, such as the infinite HMM, may pose new challenges to theoretical methodology, since their study will probably have to draw on both the theory of algorithms and mathematical statistics. Once a wider range of results is available, they may in turn serve to guide the development of new models, if it is possible to establish how different methods of model construction affect the statistical properties of the constructed model.

In addition to the references embedded in the text above, we recommend the books Hjort et al. (2010) and Ghosh and Ramamoorthi (2002), and the review articles Walker et al. (1999) and Müller and Quintana (2004) on Bayesian nonparametrics. Further references can be found in the chapter by they Teh and Jordan (2010) of the book Hjort et al. (2010).

Cross-References

▸ Bayesian Methods
▸ Dirichlet Process
▸ Gaussian Processes
▸ Mixture Modeling
▸ Prior Probability

Recommended Reading

de Finetti B (1931) Funzione caratteristica di un fenomeno aleatorio. Atti della R. Academia Nazionale dei Lincei, Serie 6. Memorie, Classe di Scienze Fisiche, Mathematice e Naturale 4:251–299

Diaconis P, Freedman D (1986) On the consistency of Bayes estimates (with discussion). Ann Stat 14(1):1–67

Dunson DB (2010) Nonparametric Bayes applications to biostatistics. In: Hjort N, Holmes C, Müller P, Walker S (eds) Bayesian nonparametrics. Cambridge University Press, Cambridge

Escobar MD, West M (1995) Bayesian density estimation and inference using mixtures. J Am Stat Assoc 90:577–588

Ghosh JK, Ramamoorthi RV (2002) Bayesian nonparametrics. Springer, New York

Hjort N, Holmes C, Müller P, Walker S (eds) (2010) Bayesian nonparametrics. Cambridge series in statistical and probabilistic mathematics, vol 28. Cambridge University Press, Cambridge

Müller P, Quintana FA (2004) Nonparametric Bayesian data analysis. Stat Sci 19(1):95–110

Neal RM (2000) Markov chain sampling methods for Dirichlet process mixture models. J Comput Graph Stat 9:249–265

Orbanz P (2010) Construction of nonparametric Bayesian models from parametric Bayes equations. In: Bengio Y, Schuurmans D, Lafferty J, Williams CKI, Culotta A (eds) Advances in neural information processing systems, Vancouver, vol 22, pp 1392–1400

Teh YW, Jordan MI (2010) Hierarchical Bayesian nonparametric models with applications. In: Hjort N, Holmes C, Müller P, Walker S (eds) Bayesian nonparametrics. Cambridge University Press, Cambridge

Walker SG, Damien P, Laud PW, Smith AFM (1999) Bayesian nonparametric inference for random distributions and related functions. J R Stat Soc 61(3):485–527

Wasserman L (2006) All of nonparametric statistics. Springer, New York

Bayesian Reinforcement Learning

Pascal Poupart
University of Waterloo, Waterloo, ON, Canada

Synonyms

Adaptive control processes; Bayes adaptive Markov decision processes; Dual control; Optimal learning

Definition

Bayesian reinforcement learning refers to ▸ reinforcement learning modeled as a Bayesian learning problem (see ▸ Bayesian Methods). More specifically, following Bayesian learning theory, reinforcement learning is performed by computing a posterior distribution on the unknowns (e.g., any combination of the transition probabilities, reward probabilities, value function, value gradient, or policy) based on the evidence received (e.g., history of past state–action pairs).

Motivation and Background

Bayesian reinforcement learning can be traced back to the 1950s and 1960s in the work of Bellman (1961), Fel'Dbaum (1965), and several of Howard's students (Martin 1967). Shortly after ▸ Markov decision processes were formalized, the above researchers (and several others) in Operations Research considered the problem of controlling a Markov process with uncertain transition and reward probabilities, which is equivalent to reinforcement learning. They considered Bayesian techniques since Bayesian learning is performed by probabilistic inference, which naturally combines with decision theory. In general, Bayesian reinforcement learning distinguishes itself from other reinforcement learning approaches by the use of probability distributions (instead of point estimates) to fully capture the uncertainty. This enables the learner to make more informed decisions, with the potential of learning faster with less data. In particular, the exploration/exploitation tradeoff can be naturally optimized. The use of a prior distribution also facilitates the encoding of domain knowledge, which is exploited in a natural and principled way by the learning process.

Structure of Learning Approach

A Markov decision process (MDP) (Puterman 1994) can be formalized by a tuple $\langle S, A, T \rangle$

where S is the set of states s, A is the set of actions a, $T(s,\ a,\ s') = \Pr(s'|s, a)$ is the transition distribution indicating the probability of reaching s' when executing a in s. Let s_r denote the reward feature of a state and $\Pr(s'_r|s, a)$ be the probability of earning r when executing a in s. A policy $\pi : S \rightarrow A$ consists of a mapping from states to actions. For a given discount factor $0 \leq \gamma \leq 1$ and horizon h, the value V^{π} of a policy π is the expected discounted total reward earned while executing this policy: $V^{\pi}(s) = \sum_{t=o}^{h} \gamma^t E_{s|\pi}[s_r^t]$. The value function V^{π} can be written in a recursive form as the expected sum of the immediate reward s'_r with the discounted future rewards: $V^{\pi}(s) = \sum_{s'} \Pr(s'|s, \pi(s))[s'_r + \gamma V \pi(s')]$. The goal is to find an optimal policy π^*, that is, a policy with the highest value V^* in all states (i.e., $V^*(s) \geq V^{\pi}(s) \forall \pi, s$). Many algorithms exploit the fact that the optimal value function V^* satisfies Bellman's equation:

$$V^*(s) = \max_a \sum_{s'} \Pr(s'|s, a)[s'_r + \gamma V^*(s)] \quad (1)$$

Reinforcement learning (Sutton and Barto 1998) is concerned with the problem of finding an optimal policy when the transition (and reward) probabilities T are unknown (or uncertain). Bayesian learning is a learning approach in which unknowns are modeled as random variables X over which distributions encode the uncertainty. The process of learning consists of updating the prior distribution $\Pr(X)$ based on some evidence e to obtain a posterior distribution $\Pr(X|e)$ according to Bayes theorem: $\Pr(X|e) = k\Pr(X)\Pr(e|X)$. (Here $k = 1/\Pr(e)$ is a normalization constant.) Hence, *Bayesian reinforcement learning* consists of using Bayesian learning for reinforcement learning. The unknowns are the transition (and reward) probabilities T, the optimal value function V^*, and the optimal policy π^*. Techniques that maintain a distribution on T are known as *model-based* Bayesian reinforcement learning since they explicitly learn the underlying model T. In contrast, techniques that maintain a distribution on V^* or π^* are known as *model-free* Bayesian reinforcement learning since they directly learn the optimal value function or policy without learning a model.

Model-Based Bayesian Learning

In model-based Bayesian reinforcement learning, the learner starts with a prior distribution over the parameters of T, which we denote by θ. For instance, let $\theta_{sas'} = \Pr(s'|s,\ a,\ \theta)$ be the unknown probability of reaching s$'$ when executing a in s. In general, we denote by θ the set of all $\theta_{sas'}$. Then, the prior $b(\theta)$ represents the initial *belief* of the learner regarding the underlying model. The learner updates its belief after every s, a, s' triple observed by computing a posterior $b_{sas'}(\theta) = b(\theta|s, a, s')$ according to Bayes theorem:

$$b_{sas'}(\theta) = kb(\theta) \Pr(s'|s, a, \theta) = kb(\theta)\theta_{sas'}. \quad (2)$$

In order to facilitate belief updates, it is convenient to pick the prior from a family of distributions that is closed under Bayes updates. This ensures that beliefs are always parameterized in the same way. Such families are called *conjugate priors*. In the case of a discrete model (i.e., $\Pr(s'|s, a, \theta)$ is a discrete distribution), Dirichlets are conjugate priors and form a family of distributions corresponding to monomials over the simplex of discrete distributions (DeGroot 1970). They are parameterized as follows: $Dir(\theta; n) = k \prod_i \theta_i^{n_i - 1}$. Here θ is an unknown discrete distribution such that $\sum_i \theta_i = 1$ and n is a vector of strictly positive real numbers n_i (known as the hyperparameters) such that $n_i - 1$ can be interpreted as the number of times that the θ_i-probability event has been observed. Since the unknown transition model θ is made up of one unknown distribution θ_a^s per s, a pair, let the prior be $b(\theta) = \prod_{s,a} Dir\ (\theta_a^s;\ n_a^s)$ such that n_a^s is a vector of hyperparameters $n_a^{s,s'}$. The posterior obtained after transition $\hat{s}, \hat{a}, \hat{s}'$ is

$$\begin{aligned} b_a^{s,s'}(\theta) &= k\theta_a^{s,s'} \prod_{s,a} Dir(\theta_a^s; n_a^s) \\ &= \prod_{s,a} Dir(\theta_a^s; n_a^s + \delta_{\hat{s},\hat{a},\hat{s}'}(s, a, s')) \end{aligned} \quad (3)$$

where $\delta_{\hat{s},\hat{a},\hat{s}'}(s, a, s')$ is a Kronecker delta that returns 1 when $s = \hat{s}, a = \hat{a}, s' = \hat{s}'$ and 0 otherwise. In practice, belief monitoring is as simple as incrementing the hyperparameter corresponding to the observed transition.

Belief MDP Equivalence

At any point in time, the belief b provides an explicit representation of the uncertainty of the learner about the underlying model. This information is very useful to decide whether future actions should focus on exploring or exploiting. Hence, in Bayesian reinforcement learning, policies π are mappings from state-belief pairs $\langle s, b \rangle$ to actions. Equivalently, the problem of Bayesian reinforcement learning can be thought as one of planning with a belief MDP (or a partially observable MDP). More precisely, every Bayesian reinforcement learning problem has an equivalent belief MDP formulation $\langle S_{bel},\ A_{bel},\ T_{bel} \rangle$ where $S_{bel} = S \times B$ (B is the space of beliefs b), $A_{bel} = A$, and $T_{bel}(s_{bel}, a_{bel}, b'_{bel}) = \Pr(b'_{bel} | b_{bel}, a_{bel}) = \Pr(s', b' | s, b, a) = \Pr(b' | s, b, a, s')\Pr(s' | s, b, a)$. The decomposition of the transition dynamics is particularly interesting since $\Pr(b' | s,\ b,\ a,\ s')$ equals 1 when $b' = b_a^{s,s'}$ (as defined in Eq. 3) and 0 otherwise. Furthermore, $\Pr(s' | s,\ b,\ a) = \int_\theta b(\theta) \Pr(s' | s, \theta, a) d\theta$, which can be computed in closed form when b is a Dirichlet. As a result, the transition dynamics of the belief MDP are fully known. This is a remarkable fact since it means that Bayesian reinforcement learning problems, which by definition have unknown/uncertain transition dynamics, can be recast as belief MDPs with known transition dynamics. While this doesn't make the problem any easier since the belief MDP has a hybrid state space (discrete s with continuous b), it allows us to treat policy optimization as a problem of planning and in particular to adapt algorithms originally designed for belief MDPs (also known as partially observable MDPs).

Optimal Value Function Parameterization

Many planning techniques compute the optimal value function V^*, from which an optimal policy π^* can easily be extracted. Despite the hybrid nature of the state space, the optimal value function (for a finite horizon) has a simple parameterization corresponding to the upper envelope of a set of polynomials (Poupart et al. 2006). Recall that the optimal value function satisfies Bellman's equation, which can be adapted as follows for a belief MDP:

$$V^*(s, b) = \max_a \sum_{s'} \Pr(s', b' | s, b, a)$$

$$[s'_r + \gamma V^*(s', b')]. \tag{4}$$

Using the fact that b' must be $b_a^{s,s'}$ (otherwise $\Pr(s',\ b' | s, b, a) = 0$) allows us to rewrite Bellman's equation as follows:

$$V^*(s, b) = \max_a \sum_{s'} \Pr(s' | s, b, a)$$

$$[s'_r + \gamma V^*(s', b_a^{s,s'})] \tag{5}$$

Let Γ^n be a set of polynomials in θ such that the optimal value function V^n with n steps to go is $V^n(s, b) = \int_\theta b(\theta) poly_{s,b}(\theta) d\theta$ where $poly_{s,b} = \operatorname{argmax}_{poly \in \Gamma_s^n} \int_\theta b(\theta) poly(\theta) d\theta$. It suffices to replace $\Pr(s' | s,\ b, a), b_a^{s,s'}$ and V^n by their definitions in Bellman's equation

$$V^{n+1}(s, b) = \max_a \sum_{s'} \int_\theta b(\theta) \Pr(s' | s, \theta, a)$$

$$[r'_s + \gamma poly_{s', b_a^{s,s'}}(\theta)] d\theta \tag{6}$$

$$= \max_a \int_\theta b(\theta) \sum_{s'} \theta_a^{s,s'}$$

$$[r'_s + \gamma poly_{s', b_a^{s,s'}}(\theta)] d\theta \tag{7}$$

to obtain a similar set of polynomials $\Gamma_s^{n+1} = \left\{ \sum_{s'} \theta_a^{s,s'} [r'_s + \gamma\ poly'_s] | a \in A, poly_{s'} \in \Gamma_{s'}^n \right\}$ that represents V^{n+1}.

The fact that the optimal value function has a closed form with a simple parameterization is quite useful for planning algorithms based on value iteration. Instead of using an arbitrary function approximator to fit the value function, one can take advantage of the fact that the value function can be represented by a set of polynomials to choose a good representation. For instance, the Beetle algorithm (Poupart et al. 2006) performs point-based value iteration and approximates the value function with a bounded set of polynomials that each consists of a linear combination of monomial basis functions.

Exploration/Exploitation Tradeoff

Since the underlying model is unknown in reinforcement learning, it is not clear whether actions should be chosen to explore (gain more information about the model) or exploit (maximize immediate rewards based on information gathered so far). Bayesian reinforcement learning provides a principled solution to the exploration/exploitation tradeoff. Despite the appearance of multiple objectives induced by exploration and exploitation, there is a single objective in reinforcement learning: maximize total discounted rewards. Hence, an optimal policy (which maximizes total discounted rewards) must naturally optimize the exploration/exploitation tradeoff. In order for a policy to be optimal, it must use all the information available. The information available to the learner consists of the history of past states and actions. One can show that state–belief pairs $\langle s, b\rangle$ are sufficient statistics of the history. Hence, by searching for the mapping from state–belief pairs to actions that maximizes total discounted rewards, Bayesian reinforcement learning implicitly seeks an optimal tradeoff between exploration and exploitation. In contrast, traditional reinforcement learning approaches search in the space of mappings from states to actions. As a result, such techniques typically focus on asymptotic convergence (i.e., convergence to a policy that is optimal in the limit), but do not effectively balance exploration and exploitation since they do not use histories or beliefs to quantify the uncertainty about the underlying model.

Related Work

Michael Duff's PhD thesis (Duff 2002) provides an excellent survey of Bayesian reinforcement learning up until 2002. The above text pertains mostly to model-based Bayesian reinforcement learning applied to discrete, fully observable, single agent domains. Bayesian learning has also been explored in model-free reinforcement learning (Dearden et al. 1998; Engel et al. 2005; Ghavamzadeh and Engel 2006) continuous-valued state spaces (Ross et al. 2008), partially observable domains (Poupart and Vlassis 2008; Ross et al. 2007), and multi-agent systems (Chalkiadakis and Boutilier 2003, 2004; Gmytrasiewicz and Doshi 2005).

Cross-References

- ▸ Active Learning
- ▸ Markov Decision Processes
- ▸ Reinforcement Learning

Recommended Reading

Bellman R (1961) Adaptive control processes: a guided tour. Princeton University Press, Princeton

Chalkiadakis G, Boutilier C (2003) Coordination in multi-agent reinforcement learning: a Bayesian approach. In: International joint conference on autonomous agents and multiagent systems (AAMAS), Melbourne, pp 709–716

Chalkiadakis G, Boutilier C (2004) Bayesian reinforcement learning for coalition formation under uncertainty. In: International joint conference on autonomous agents and multiagent systems (AAMAS), New York, pp 1090–1097

Dearden R, Friedman N, Russell SJ (1998) Bayesian Q-learning. In: National conference on artificial intelligence (AAAI), Madison, pp 761–768

DeGroot MH (1970) Optimal statistical decisions. McGraw-Hill, New York

Duff M (2002) Optimal learning: computational procedures for Bayes-adaptive Markov decision processes. PhD thesis, University of Massachusetts, Amherst

Engel Y, Mannor S, Meir R (2005) Reinforcement learning with Gaussian processes. In: International conference on machine learning (ICML), Bonn

Fel'Dbaum A (1965) Optimal control systems. Academic, New York

Ghavamzadeh M, Engel Y (2006) Bayesian policy gradient algorithms. In: Advances in neural information processing systems (NIPS), Vancouver, pp 457–464

Gmytrasiewicz P, Doshi P (2005) A framework for sequential planning in multi-agent settings. J Artif Intell Res (JAIR) 24:49–79

Martin JJ(1967) Bayesian decision problems and Markov chains. Wiley, New York

Poupart P, Vlassis N (2008) Model-based Bayesian reinforcement learning in partially observable domains. In: International symposium on artificial intelligence and mathematics (ISAIM), Beijing

Poupart P, Vlassis N, Hoey J, Regan K (2006) An analytic solution to discrete Bayesian reinforcement learning. In: International conference on machine learning (ICML), Pittsburgh, pp 697–704
Puterman ML (1994) Markov decision processes. Wiley, New York
Ross S, Chaib-Draa B, Pineau J (2007) Bayes-adaptive POMDPs. In: Advances in neural information processing systems (NIPS), Vancouver
Ross S, Chaib-Draa B, Pineau J (2008) Bayesian reinforcement learning in continuous POMDPs with application to robot navigation. In: IEEE international conference on robotics and automation (ICRA), Pasadena, pp 2845–2851
Sutton RS, Barto AG (1998) Reinforcement learning. MIT Press, Cambridge, MA

Beam Search

Claude Sammut
The University of New South Wales, Sydney, NSW, Australia

A beam search is a heuristic search technique that combines elements of breadth-first and best-first searches. Like a breadth-first search, the beam search maintains a list of nodes that represent a frontier in the search space. Whereas the breadth-first adds all neighbors to the list, the beam search orders the neighboring nodes according to some heuristic and only keeps the n best, where n is the *beam size*. This can significantly reduce the processing and storage requirements for the search.

In machine learning, the beam search has been used in algorithms, such as Dietterich and Michalski (1977).

Cross-References

▸ Learning as Search

Recommended Reading

Dietterich TG, Michalski RS (1977) Learning and generalization of characteristic descriptions: evaluation criteria and comparative review of selected methods. In: Fifth international joint conference on artificial intelligence, pp 223–231. William Kaufmann, Cambridge

Behavioral Cloning

Claude Sammut
The University of New South Wales, Sydney, NSW, Australia

Synonyms

Apprenticeship learning; Learning by demonstration; Learning by imitation; Learning control rules

Definition

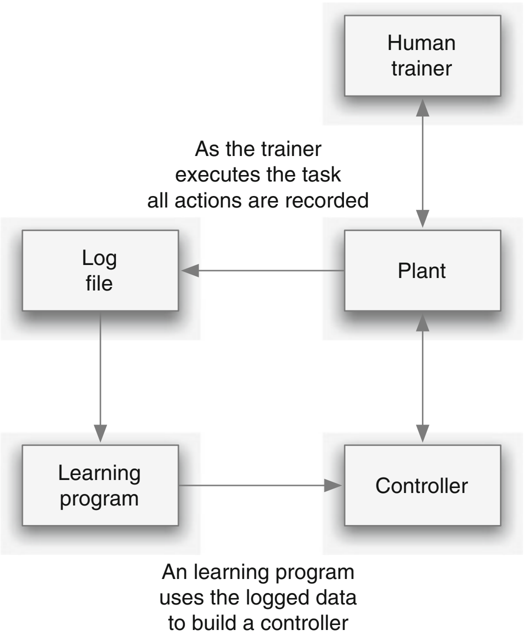
Behavioral cloning is a method by which human subcognitive skills can be captured and reproduced in a computer program. As the human subject performs the skill, his or her actions are recorded along with the situation that gave rise to the action. A log of these records is used as input to a learning program. The learning program outputs a set of rules that reproduce the skilled behavior. This method can be used to construct automatic control systems for complex tasks for which classical control theory is inadequate. It can also be used for training.

Motivation and Background

Behavioral cloning (Michie et al. 1990) is a form of *learning by imitation* whose main motivation is to build a model of the behavior of a human when performing a complex skill. Preferably, the model should be in a readable form. It is related to other forms of learning by imitation, such as ▸ inverse reinforcement learning (Abbeel and Ng 2004; Amit and Matariæ 2002; Hayes and Demiris 1994; Kuniyoshi et al. 1994; Pomerleau 1989) and methods that use data from human performances to model the system being controlled (Atkeson and Schaal 1997; Bagnell and Schneider 2001).

Experts might be defined as people who know what they are doing not what they are talking about. That is, once a person becomes highly skilled in some task, the skill becomes sub-cognitive and is no longer available to introspection. So when the person is asked to explain why certain decisions were made, the explanation is a post hoc justification rather than a true explanation.

Michie et al. (1990) used an induction program to learn rules for balancing a pole (in simulation) and earlier work by Donaldson (1960), Widrow and Smith (1964), and Chambers and Michie (1969) demonstrated the feasibility of learning by imitation, also for pole-balancing.



As the trainer executes the task all actions are recorded

An learning program uses the logged data to build a controller

Behavioral Cloning, Fig. 1 Structure of learning system

Structure of the Learning System

Behavioral cloning assumes that there is a plant of some kind that is under the control of a human operator. The plant may be a physical system or a simulation. In either case, the plant must be instrumented so that it is possible to capture the state of the system, including all the control settings. Thus, whenever the operator performs an action, that is, changes a control setting, we can associate that action with a particular state.

Let us use a simple example of a system that has only one control action. A *pole balancer* has four state variables: the angle of the pole, θ, and its angular velocity, $\dot{\theta}$ and the position, x, and velocity $\dot{x}$, of the cart on the track. The only action available to the controller is to apply a fixed positive of negative force, F, to accelerate the cart left or right (Fig. 1).

We can create an experimental setup where a human can control a pole and cart system (either real or in simulation) by applying a left push or a right push at the appropriate time. Whenever a control action is performed, we record the action as well as values of the four state variables at the time of the action. Each of these records can be viewed as an example of a mapping from state to action.

Michie et al. (1990) demonstrated that it is possible to construct a controller by learning from these examples. The learning task is to predict the appropriate action, given the state. They used a ▸ decision tree learning program to produce a classifier that, given the values of the four state variables, would output an action. A decision tree is easily convertible into an executable code as a nested **if** statement. The quality of the controller can be tested by inserting the decision tree into the simulator, replacing the human operator.

If the goal of learning is simply to produce an operational controller then any program capable of building a classifier could be used. The reason that Michie et al. (1990) chose a symbolic learner was their desire to produce a controller whose decision making was transparent as well as operational. That is, it should be possible to extract an explanation of the behavior that is meaningful to an expert in the task.

Learning Direct (Situation–Action) Controllers

A controller such as the one described above is referred to as a *direct controller* because it maps situations to actions. Other examples of learning a direct controller are building an autopilot from behavioral traces of human pilots flying aircraft in a flight simulator (Sammut et al. 1992) and building a control system for a container crane

(Urbani and Bratko 1994). These systems extended the earlier work by operating in domains in which there is more than one control variable and the task is sufficiently complex that it must be decomposed into several subtasks.

An operator of a container crane can control the speed of the cart and the length of the rope. A pilot of a fixed-wing aircraft can control the ailerons, elevators, rudder, throttle, and flaps. To build an autopilot, the learner must build a system that can set each of the control variables. Sammut et al. (1992), viewed this as a multitask learning problem.

Each training example is a feature vector that includes the position, orientation, and velocities of the aircraft as well as the values of each of the control settings: ailerons, elevator, throttle, and flaps. The rudder is ignored. A separate decision tree is built for each control variable. For example, the aileron setting is treated as the dependent variable and all the other variables, including the other controls, are treated as the attributes of the training example. A decision tree is built for ailerons, then the process is repeated for the elevators, etc. The result is a decision tree for each control variable.

The autopilot code executes each decision tree in each cycle of the control loop. This method treats the setting of each control as a separate task. It may be surprising that this method works since it is often necessary to adjust more than one control simultaneously to achieve the desired result. For example, to turn, it is normal to use the ailerons to roll the aircraft while adjusting the elevators to pull it around. This kind of multivariable control does result from multiple decision trees. When, say, the aileron decision tree initiates a roll, the elevator's decision tree detects the roll and causes the aircraft to pitch up and execute a turn.

Limitations

Direct controllers work quite well for systems that have a relatively small state space. However, for complex systems, behavioral cloning of direct situation–action rules tends to produce very brittle controllers. That is, they cannot tolerate large disturbances. For example, when air turbulence is introduced into the flight simulator, the performance of the clone degrades very rapidly. This is because the examples provided by logging the performance of a human only cover a very small part of the state space of a complex system such as an aircraft in flight. Thus, the"expertise" of the controller is very limited. If the system strays outside the controller's region of expertise, it has no method for recovering and failure is usually catastrophic.

More robust control is possible but only with a significant change in approach. The more successful methods decompose the learning task into two stages: learning goals and learning the actions to achieve those goals.

Learning Indirect (Goal-Directed) Controllers

The problem of learning in a large search space can partially be addressed by decomposing the learning into subtasks. A controller built in this way is said to be an *indirect controller*. A control is "indirect" if it does not compute the next action directly from the system's current state but uses, in addition, some intermediate information. An example of such intermediate information is a subgoal to be attained before achieving the final goal.

Subgoals often feature in an operator's control strategies and can be automatically detected from a trace of the operator's behavior (Šuc and Bratko 1997). The problem of subgoal identification can be treated as the inverse of the usual problem of controller design, that is, given the actions in an operator's trace, find the goal that these actions achieve. The limitation of this approach is that it only works well for cases in which there are just a few subgoals, not when the operator's trajectory contains many subgoals. In these cases, a better approach is to generalize the operator's trajectory. The generalized trajectory can be viewed as defining a continuously changing subgoal (Bratko and Šuc 2002; Šuc and Bratko 1999a) (see also the use of flow tubes in dynamic plan execution Hofmann and Williams 2006).

Subgoals and generalized trajectories are not sufficient to define a controller. A model of the systems dynamics is also required. Therefore, in addition to inducing subgoals or a generalized trajectory, this approach also requires learning approximate system dynamics, that is a model of the controlled system. Bratko and Šuc (2003) and use a combination of the Goldhorn (Križman and Džeroski 1995) discovery program and locally weighted regression to build the model of the system's dynamics. The next action is then computed "indirectly" by (1) computing the desired next state (e.g., next subgoal) and (2) determining an action that brings the system to the desired next state. Bratko and Šuc also investigated building qualitative control strategies from operator traces (Bratko and Šuc 2002).

An analog to this approach is ▸ inverse reinforcement learning (Abbeel and Ng 2004; Atkeson and Schaal 1997; Ng and Russell 2000) where the reward function is learned. Here, the learning the reward function corresponds to learning the human operator's goals.

Isaac and Sammut (2003) uses an approach that is similar in spirit to Šuc and Bratko but incorporates classical control theory. Learned skills are represented by a two-level hierarchical decomposition with an anticipatory goal level and a reactive control level. The goal level models how the operator chooses goal settings for the control strategy and the control level models the operator's reaction to any error between the goal setting and actual state of the system. For example, in flying, the pilot can achieve goal values for the desired heading, altitude, and airspeed by choosing appropriate values of turn rate, climb rate, and acceleration. The controls can be set to correct errors between the current state and the desired state of these goal-directing quantities. Goal models map system states to a goal setting. Control actions are based on the error between the output of each of the goal models and the current system state.

The control level is modeled as a set of proportional integral derivative (PID) controllers, one for each control variable. A PID controller determines a control value as a linear function proportional to the error on a goal variable, the integral of the error, and the derivative of the error.

Goal setting and control models are learned separately. The process begins be deciding which variables are to be used for the goal settings. For example, trainee pilots will learn to execute a "constant-rate turn," that is, their goal is to maintain a given turn rate. A separate goal rule is constructed for each goal variable using a ▸ model tree learner (Potts and Sammut 2005).

A goal rule gives the setting for a goal variable and therefore, we can find the difference (error) between the current state value and the goal setting. The integral and derivative of the error can also be calculated. For example, if the set turn rate is 180° min, then the error on the turn rate is calculated as the actual turn rate minus 180. The integral is then the running sum of the error multiplied by the time interval between time samples, starting from the first time sample of the behavioral trace, and the derivative is calculated as the difference between the error and previous error all divided by the time interval.

For each control available to the operator, a model tree learner is used to predict the appropriate control setting. ▸ Linear regression is used in the leaf nodes of the model tree to produce linear equations whose coefficients are the P, I, and D of values of the PID controller. Thus the learner produces a collection of PID controllers that are selected according to the conditions in the internal nodes of the tree. In control theory, this is known as *piecewise linear control*.

Another indirect method is to learn a model of the dynamics of the system and use this to learn, in simulation, a controller for the system (Bagnell and Schneider 2001; Ng et al. 2003). This approach does not seek to directly model the behavior of a human operator. A behavioral trace may be used to generate data for modeling the system but then a reinforcement learning algorithm is used to generate a policy for controlling the simulated system. The learned policy can then be transferred to the physical system. ▸ Locally weighted regression is typically used for system modeling, although ▸ model trees can also be used.

Cross-References

- ▸ Apprenticeship Learning
- ▸ Inverse Reinforcement Learning
- ▸ Learning by Imitation
- ▸ Locally Weighted Regression for Control
- ▸ Model Trees
- ▸ Reinforcement Learning

Recommended Reading

Abbeel P, Ng AY (2004) Apprenticeship learning via inverse reinforcement learning. In: International conference on machine learning, Banff. ACM, New York

Amit R, Matariæ M (2002) Learning movement sequences from demonstration. In: Proceedings of the second international conference on development and learning, Cambridge, MA. IEEE, Washington, DC, pp 203–208

Atkeson CG, Schaal S (1997) Robot learning from demonstration. In: Fisher DH (ed) Proceedings of the fourteenth international conference on machine learning, Nashville. Morgan Kaufmann, San Francisco, pp 12–20

Bagnell JA, Schneider JG (2001) Autonomous helicopter control using reinforcement learning policy search methods. In: International conference on robotics and automation, Seoul. IEEE Press, New York

Bratko I, Šuc D (2002) Using machine learning to understand operator's skill. In: Proceedings of the 15th international conference on industrial and engineering applications of artificial intelligence and expert systems. Springer/AAAI Press, London/Menlo Park, pp 812–823

Bratko I, Šuc D (2003) Learning qualitative models. AI Mag 24(4):107–119

Chambers RA, Michie D (1969) Man-machine cooperation on a learning task. In: Parslow R, Prowse R, Elliott-Green R (eds) Computer graphics: techniques and applications. Plenum, London

Donaldson PEK (1960) Error decorrelation: a technique for matching a class of functions. In: Proceedings of the third international conference on medical electronics, London, pp 173–178

Hayes G, Demiris J (1994) A robot controller using learning by imitation. In: Proceedings of the international symposium on intelligent robotic systems, Grenoble. LIFTA-IMAG, Grenoble, pp 198–204

Hofmann AG, Williams BC (2006) Exploiting spatial and temporal flexiblity for plan execution of hybrid, under-actuated systems. In: Proceedings of the 21st national conference on artficial intelligence, Boston, pp 948–955

Isaac A, Sammut C (2003) Goal-directed learning to fly. In: Fawcett T, Mishra N (eds) Proceedings of the twentieth international conference on machine learning, Washington, DC. AAAI, Menlo Park, pp 258–265

Križman V, Džeroski S (1995) Discovering dynamics from measured data. Electrotech Rev 62(3–4):191–198

Kuniyoshi Y, Inaba M, Inoue H (1994) Learning by watching: extracting reusable task knowledge from visual observation of human performance. IEEE Trans Robot Autom 10:799–822

Michie D, Bain M, Hayes-Michie JE (1990) Cognitive models from subcognitive skills. In: Grimble M, McGhee S, Mowforth P (eds) Knowledge-based systems in industrial control. Peter Peregrinus, Stevenage

Ng AY, Jin Kim H, Jordan MI, Sastry S (2003) Autonomous helicopter flight via reinforcement learning. In: Thrun S, Saul L, Schölkopf B (eds) Advances in neural information processing systems, vol 16. MIT Press, Cambridge

Ng AY, Russell S (2000) Algorithms for inverse reinforcement learning. In: Proceedings of 17th international conference on machine learning, Stanford. Morgan Kaufmann, San Francisco, pp 663–670

Pomerleau DA (1989) ALVINN: an autonomous land vehicle in a neural network. In: Touretzky DS (ed) Advances in neural information processing systems. Morgan Kaufmann, San Mateo

Potts D, Sammut C (2005) Incremental learning of linear model trees. Mach Learn 6(1–3):5–48

Sammut C, Hurst S, Kedzier D, Michie D (1992) Learning to fly. In: Sleeman D, Edwards P (eds) Proceedings of the ninth international conference on machine learning, Aberdeen. Morgan Kaufmann, San Francisco, pp 385–393

Šuc D, Bratko I (1997) Skill reconstruction as induction of LQ controllers with subgoals. In: IJCAI-97: proceedings of the fifteenth international joint conference on artificial intelligence, Nagoya, vol 2. Morgan Kaufmann, San Francisco, pp 914–920

Šuc D, Bratko I (1999a) Modelling of control skill by qualitative constraints. In: Thirteenth international workshop on qualitative reasoning, Lock Awe. University of Aberystwyth, Aberystwyth, pp 212–220

Šuc D, Bratko I (1999b) Symbolic and qualitative reconstruction of control skill. Electron Trans Artif Intell 3(B):1–22

Urbančič T, Bratko I (1994) Reconstructing human skill with machine learning. In: Cohn A (ed) Proceedings of the 11th European conference on artificial intelligence. Wiley, Amsterdam/New York

Widrow B, Smith FW (1964) Pattern recognising control systems. In: Tou JT, Wilcox RH (eds) Computer and information sciences. Clever Hume, London

Belief State Markov Decision Processes

▸ Partially Observable Markov Decision Processes

Bellman Equation

The *Bellman Equation* is a recursive formula that forms the basis for ▸ dynamic programming. It computes the expected total reward of taking an action from a state in a ▸ Markov decision process by breaking it into the immediate reward and the total future expected reward. See ▸ dynamic programming.

Bias

Bias has two meanings, ▸ inductive bias, and *statistical bias* see ▸ bias variance decomposition.

Bias Specification Language

Hendrik Blockeel
Katholieke Universiteit Leuven, Heverlee,
Leuven, Belgium
Leiden Institute of Advanced Computer Science,
Heverlee, Belgium

Definition

A *bias specification language* is a language in which a user can specify a ▸ Language Bias. The language bias of a learner is the set of hypotheses (or hypothesis descriptions) that this learner may return.

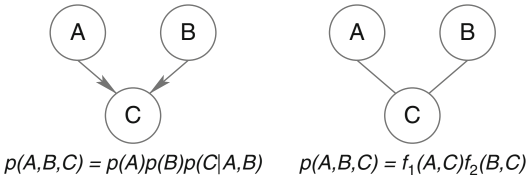
In contrast to the ▸ hypothesis language, the bias specification language allows us to describe not single hypotheses but sets (languages) of hypotheses.

Examples

In learning approaches based on ▸ graphical models or ▸ artificial neural networks, whenever the user provides the graph structure of the model, he or she is specifying a bias. The "language" used to specify this bias, in this case, consists of graphs. Figure 1 shows examples of such graphs. Not every kind of bias can necessarily be expressed by some bias specification language; for instance, the bias defined by the ▸ Bayesian network structure in Fig. 1 cannot be expressed using a ▸ Markov network. Bayesian networks and Markov networks have a different expressiveness, when viewed as bias specification languages.

Also certain parameters of decision tree learners or rule set learners effectively restrict the hypothesis language (for instance, an upper bound on the rule length or the size of the decision tree).

A combination of parameter values can hardly be called a language, and even the "language" of graphs is a relatively simple kind of language. More elaborate types of bias specification languages are typically found in the field of ▸ inductive logic programming (ILP).



Bias Specification Language, Fig. 1 Graphs defining a bias for learning joint distributions. The Bayesian network structure to the left constrains the form of the joint distribution in a particular way (shown as the equation below the graph). Notably, it guarantees that only distributions can be learned in which the variables A and B are (unconditionally) independent. The Markov network structure to the right constrains the form of the joint distribution in a different way: it states that it must be possible to write the distribution as a product of a function of A and C and a function of B and C. These two biases are different. In fact, no Markov network structure over the variables A, B, and C exists that expresses the bias specified by the Bayesian network structure

Bias Specification Languages in Inductive Logic Programming

In ILP, the hypotheses returned by the learning algorithm are typically written as first-order logic clauses. As the set of all possible clauses is too large to handle, a subset of these clauses is typically defined; this subset is called the language bias. Several formalisms (“bias specification languages”) have been proposed for specifying such subsets. We here focus on a few representative ones.

DLAB

In the DLAB bias specification language (Dehaspe and De Raedt 1996), the language bias is defined in a declarative way, by defining a syntax that clauses must fulfill. In its simplest form, a DLAB specification simply gives a set of possible head and body literals out of which the system can build a clause.

Example 1 The actual syntax of the DLAB specification language is relatively complicated (see Dehaspe and De Raedt 1996), but in essence, one can write down a specification such as:

```
{ parent({X,Y,Z},{X,Y,Z}),
   grandparent({X,Y,Z},
      {X,Y,Z})}
:-
{ parent({X,Y,Z},{X,Y,Z}),
   parent({X,Y,Z},{X,Y,Z}),
   grandparent({X,Y,Z},{X,Y,Z}),
   grandparent({X,Y,Z}, {X,Y,Z}) }
```

which states that the hypothesis language consists of all clauses that have at most one parent and at most one grandparent literal in the head, and at most two of these literals in the body; the arguments of these literals may be variables X,Y,Z. Thus, the following clauses are in the hypothesis language:

```
grandparent(X, Y)
:- parent(X, Z), parent(Z,Y)
:- parent(X,Y), parent(Y,X)
:- parent(X,X)
```

These express the usual definition of grandparent as well as the fact that there can be no cycles in the parent relation.

Note that for each argument of each literal, all the variables and constants that may occur have to be enumerated explicitly. This can make a DLAB specification quite complex. While DLAB contains advanced constructs to alleviate this problem, it remains the case that often very elaborate bias specifications are needed in practical situations.

Type-and Mode-Based Biases

A more flexible bias specification language is used by Progol (Muggleton 1995) and many other ILP systems. It is based on the notions of types and modes. In Progol, arguments of a predicate can be typed, and a variable can never occur in two locations with different types. Similarly, arguments of a predicate have an input (+) or output (−) mode; each variable that occurs as an input argument of some literal must occur elsewhere as an output argument, or must occur as input argument in the head literal of a clause.

Example 2 In Progol, the specifications

```
type(parent(human,human)).
type(grandparent(human,human)).
modeh(grandparent(+,+)).
   % modeh: specifies a head
literal modeb(grandparent(+,-)).
   % modeb: specifies a body
literal modeb(parent(+,-)).
```

allow the system to construct a clause such as

```
grandparent(X,Y) :- parent(X,Z),
     parent(Z,Y)
```

but not the following clause:

```
grandparent(X,Y) :- parent(Z,Y)
```

because Z occurs as an input parameter for parent without occurring elsewhere as an output parameter (i.e., it is being used without having been given a value first).

FLIPPER's Bias Specification Language

The FLIPPER system (Cohen 1996) employs a powerful, but somewhat more procedural, bias specification formalism. The user does not specify a set of valid hypotheses directly, but rather, specifies a Refinement Operator. The language bias is the set of all clauses that can be obtained from one or more starting clauses through repeated application of this refinement operator. The operator itself is defined by specifying under which conditions certain literals can be added to a clause.

Rules defining the operator have one of the following forms:

- $A \leftarrow B$ where *Pre* asserting *Post*
- L where *Pre* asserting *Post*

The first form defines a set of "starting clauses," and the second form defines when a literal L can be added to a clause. Each rule can only be applied when its preconditions *Pre* are fulfilled, and upon application will assert a set of literals *Post*. As an example (Cohen 1996), the rules

$illegal(A, B, C, D, E, F) \leftarrow$

`where` *true*
`asserting` $\{linked(A), linked(B), \ldots,$ $linked(F)\}$

$R(X, Y)$ where $rel(R)$, $linked(X)$, $linked(Y)$
`asserting` Ø

state that any clause of the form

illegal (A, B,C,D, E, F)←

can be used as a starting point for the refinement operator, and the variables in this clause are all *linked*. Further, any literal of the form $R(X, Y)$ with R a relation symbol (as defined by the *Rel* predicate) and X and Y linked variables can be added.

Other Approaches

Grammars or term rewriting systems have been proposed several times as a means of defining the hypothesis language. A relatively recent approach along these lines was given by Lloyd, who uses a rewriting system to define the tests that can occur in the nodes of a decision tree built by the Alkemy system (Lloyd 2003).

Boström and Idestam-Almquist (1999) present an approach where the language bias is implicitly defined through the Background Knowledge given to the learner.

Knobbe et al. (2000) propose the use of UML as a "common" bias specification language, specifications in which could be translated automatically to languages specific to a particular learner.

Further Reading

An overview of bias specification formalisms in ILP is given by Nédellec et al. (1996). The bias specification languages discussed above are discussed in more detail in Dehaspe and De Raedt (1996), Muggleton (1995), and Cohen (1996). De Raedt (1992) discusses language bias and the concept of bias shift (learners weakening their bias, i.e., extending the set of hypotheses that can be represented, when a given language bias turns out to be too restrictive). A more recent approach to learning declarative bias is presented by Bridewell and Todorovski (2008).

Cross-References

▶ Hypothesis Language
▶ Inductive Logic Programming

Recommended Reading

Boström H, Idestam-Almquist P (1999) Induction of logic programs by example-guided unfolding. J Log Program 40(2–3):159–183

Bridewell W, Todorovski L (2008) Learning declarative bias. In: Proceedings of the 17th international conference on inductive logic programming. Lecture notes in computer science, vol 4894. Springer, Berlin, pp 63–77

Cohen W (1996) Learning to classify English text with ILP methods. In: De Raedt L (ed) Advances in inductive logic programming. IOS Press, Amsterdam, pp 124–143

De Raedt L (1992) Interactive theory revision: an inductive logic programming approach. Academic Press, New York

Dehaspe L, De Raedt L (1996) DLAB: a declarative language bias formalism. In: Proceedings of the international symposium on methodologies for intelligent systems. Lecture notes in artificial intelligence, vol 1079. Springer, Berlin, pp 613–622

Knobbe AJ, Siebes A, Blockeel H, van der Wallen D (2000) Multi-relational data mining, using UML for ILP. In: Proceedings of PKDD-2000 – the fourth European conference on principles and practice of knowledge discovery in databases. Lecture notes in artificial intelligence, Lyon, vol 1910. Springer, Berlin, pp 1–12

Lloyd JW (2003) Logic for learning. Springer, Berlin

Muggleton S (1995) Inverse entailment and Progol. New Gener Comput Spec Issue Inductive Log Program 13(3–4):245–286

Nédellec C, Adé H, Bergadano F, Tausend B (1996) Declarative bias in ILP. In: De Raedt L (ed) Advances in inductive logic programming. Frontiers in artificial intelligence and applications, vol 32. IOS Press, Amsterdam, pp 82–103

Bias Variance Decomposition

Definition

The bias-variance decomposition is a useful theoretical tool to understand the performance characteristics of a learning algorithm. The following discussion is restricted to the use of *squared loss* as the performance measure, although similar analyses have been undertaken for other loss functions. The case receiving most attention is the zero-one loss (i.e., classification problems), in which case the decomposition is nonunique and a topic of active research. See Domingos (1992) for details.

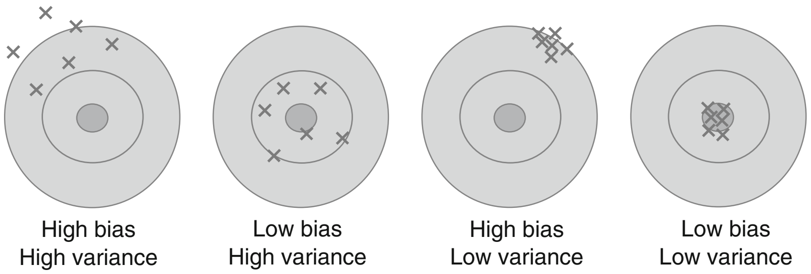
The decomposition allows us to see that the mean squared error of a model (generated by a particular learning algorithm) is in fact made up of two components. The *bias* component tells us how accurate the model is, on average across different possible training sets. The *variance* component tells us how sensitive the learning algorithm is to small changes in the training set (Fig. 1).

Mathematically, this can be quantified as a decomposition of the mean squared error function. For a testing example $\{\mathbf{x}, d\}$, the decomposition is:

$$\mathcal{E}_{\mathcal{D}}\{(f(\mathbf{x})-d)^2\} = (\mathcal{E}_{\mathcal{D}}\{f(\mathbf{x})\}-d)^2 + \mathcal{E}_{\mathcal{D}}\{(f(\mathbf{x})-\mathcal{E}_{\mathcal{D}}\{f(\mathbf{x})\})^2\},$$

$$\text{MSE} = \text{bias}^2 + \text{variance},$$

where the expectations are with respect to all possible training sets. In practice, this can be estimated by cross-validation over a single finite training set, enabling a deeper understanding of the algorithm characteristics. For example, efforts to reduce variance often cause increases in bias, and vice versa. A large bias and low variance is an indicator that a learning algorithm is prone to ▶ overfitting the model.



Bias Variance Decomposition, Fig. 1 The bias-variance decomposition is like trying to hit the bullseye on a dartboard. Each dart is thrown after training our "dart-throwing" model in a slightly different manner. If the darts vary wildly, the learner is *high variance*. If they are far from the bullseye, the learner is *high bias*. The ideal is clearly to have both low bias and low variance; however this is often difficult, giving an alternative terminology as the bias-variance "dilemma" (*Dartboard analogy*, Moore and McCabe 2002)

Cross-References

▶ Bias-Variance Trade-Offs: Novel Applications

Recommended Reading

Domingos P (1992) A unified bias-variance decomposition for zero-one and squared loss. In: Proceedings of national conference on artificial intelligence. AAAI Press, Austin

Geman S (1992) Neural networks and the bias/variance dilemma. Neural Comput 4(1):1–58

Moore DS, McCabe GP (2002) Introduction to the practice of statistics. Michelle Julet

Bias-Variance Trade-Offs: Novel Applications

Dev Rajnarayan[1] and David Wolpert[1,2]
[1]NASA Ames Research Center, Moffett Field, CA, USA
[2]Santa Fe Institute, Santa Fe, NM, USA

Definition

Consider a given random variable $\underline{F}$ and a random variable that we can modify, $\hat{\underline{F}}$. We wish to use a sample of $\hat{\underline{F}}$ as an estimate of a sample of $\underline{F}$. The mean squared error (MSE) between such a pair of samples is a sum of four terms. The first term reflects the statistical coupling between $\underline{F}$ and $\hat{\underline{F}}$ and is conventionally ignored in bias-variance analysis. The second term reflects the inherent noise in $\underline{F}$ and is independent of the estimator $\hat{\underline{F}}$. Accordingly, we cannot affect this term. In contrast, the third and fourth terms depend on $\hat{\underline{F}}$. The third term, called the bias, is independent of the precise samples of both $\underline{F}$ and $\hat{\underline{F}}$ and reflects the difference between the means of $\underline{F}$ and $\hat{\underline{F}}$. The fourth term, called the variance, is independent of the precise sample of $\underline{F}$ and reflects the inherent noise in the estimator as one samples it. These last two terms can be modified by changing the choice of the estimator. In particular, on small sample sets, we can often decrease our mean squared error by, for instance, introducing a small bias that causes a large reduction the variance. While most commonly used in machine learning, this article shows that such bias-variance trade-offs are applicable in a much broader context and in a variety of situations. We also show, using experiments, how existing bias-variance trade-offs can be applied in novel circumstances to improve the performance of a class of optimization algorithms.

Motivation and Background

In its simplest form, the bias-variance decomposition is based on the following idea. Say we have a random variable $\underline{F}$ taking on values F distributed according to a density function $p(F)$. We want to estimate the value of a sample from $p(F)$. To form our estimate, we sample a different random variable $\hat{\underline{F}}$ taking on values $\hat{F}$ distributed according to $p(\hat{F})$. Assuming a quadratic loss function, the quality of our estimate is measured by its MSE:

$$\mathrm{MSE}(\hat{\underline{F}}) \equiv \int p(\hat{F}, F)\,(\hat{F} - F)^2 \mathrm{d}\hat{F}\,\mathrm{d}F.$$

In many situations, $\underline{F}$ and $\hat{\underline{F}}$ are dependent variables. For example, in supervised machine learning, $\underline{F}$ is a "target" conditional distribution, stochastically mapping elements of an input space X into a space Y of output variables. The associated distribution $p(F)$ is the "prior" of $\underline{F}$. A random sample $\mathcal{D}$ of $\underline{F}$, called "the training set," is generated, and $\mathcal{D}$ is used in a "learning algorithm" to produce $\hat{\underline{F}}$, which is our estimate of $\underline{F}$. Clearly, this $\underline{F}$ and $\hat{\underline{F}}$ are statistically dependent, via $\mathcal{D}$. Indeed, intuitively speaking, the goal in designing a learning algorithm is that the $\hat{\underline{F}}$'s it produces are positively correlated with $\underline{F}$'s.

In practice this coupling is simply ignored in analyses of bias plus variance, without any justification (one such justification could be that the coupling has little effect on the value of the MSE). We shall follow that practice here. Accordingly, our equation for MSE reduces to

$$\mathrm{MSE}(\underline{\hat{F}}) = \int p(\hat{F})p(F)\,(\hat{F}-F)^2 \mathrm{d}\hat{F}\,\mathrm{d}F. \tag{1}$$

If we were to account for the coupling of $\hat{F}$ and $\underline{\hat{F}}$, an additive correction term would need to be added to the right-hand side. For instance, see Wolpert (1997).

Using simple algebra, the right-hand side of (1) can be written as the sum of three terms. The first is the variance of $\underline{F}$. Since this is beyond our control in designing the estimator $\underline{\hat{F}}$, we ignore it for the rest of this article. The second term involves a mean that describes the deterministic component of the error. This term depends on both the distribution of $\underline{F}$ and that of $\underline{\hat{F}}$ and quantifies how close the means of those distributions are. The third term is a variance that describes stochastic variations from one sample to the next. This term is independent of the random variable being estimated. Formally, up to an overall additive constant, we can write

$$\begin{aligned}
\mathrm{MSE}(\hat{F}) &= \int p(\hat{F})(\hat{F}^2 - 2F\hat{F} + F^2)\,\mathrm{d}\hat{F} \\
&= \int p(\hat{F})\hat{F}^2\,\mathrm{d}\hat{F} \\
&\quad -2F\int p(\hat{F})\hat{F}\,\mathrm{d}\hat{F} + F^2 \\
&= \overbrace{\mathbb{V}(\hat{F}) + [\mathbb{E}(\hat{F})]^2} - 2F\,\mathbb{E}(\hat{F}) + F^2 \\
&= \mathbb{V}(\hat{F}) + \underbrace{[F - \mathbb{E}(\hat{F})]^2} \\
&= \text{variance} + \text{bias}^2.
\end{aligned} \tag{2}$$

In light of (2), one way to try to reduce expected quadratic error is to modify an estimator to trade-off bias and variance. Some of the most famous applications of such bias-variance trade-offs occur in parametric machine learning, where many techniques have been developed to exploit the trade-off. Nonetheless, the trade-off also arises in many other fields, including integral estimation and optimization. In the rest of this paper, we present a few novel applications of bias-variance trade-off and describe some interesting features in each case. A recurring theme is the following: whenever a bias-variance trade-off arises in a particular field, we can use many techniques from parametric machine learning that have been developed for exploiting this trade-off. See Wolpert and Rajnarayan (2007) for further details of many of these applications.

Applications

In this section, we describe some applications of the bias-variance trade-off. First, we describe Monte Carlo (MC) techniques for the estimation of integrals and provide a brief analysis of bias-variance trade-offs in this context. Next, we introduce the field of Monte Carlo optimization (MCO) and illustrate that there are more subtleties involved than in simple MC. Then, we describe the field of parametric machine learning, which, as will show, is formally identical to MCO. Finally, we describe the application of parametric learning (PL) techniques to improve the performance of MCO algorithms. We do this in the context of an MCO problem that addresses black-box optimization.

Monte Carlo Estimation of Integrals Using Importance Sampling

Monte Carlo methods are often the method of choice for estimating difficult high-dimensional integrals. Consider a function $f : X \to \mathbb{R}$, which we want to integrate over some region $\mathcal{X} \subseteq X$, yielding the value F, as given by

$$F = \int_{\mathcal{X}} \mathrm{d}x\; f(x).$$

We can view this as a random variable $\underline{F}$, with density function given by a Dirac delta function centered on F. Therefore, the variance of $\underline{F}$ is 0, and (2) is exact.

A popular MC method to estimate this integral is importance sampling (see Robert and Casella 2004). This exploits the law of large numbers as follows: i.i.d. samples $x^{(i)}, i = 1, \ldots, m$ are generated from a so-called importance distribution $h(x)$ that we control, and the associated values of the integrand $f(x^{(i)})$ are computed. Denote these "data" by

$$\mathcal{D} = \left\{(x^{(i)}, f(x^{(i)}), \; i = 1, \ldots, m\right\}. \qquad (3)$$

Now,

$$\begin{aligned} F &= \int_{\mathcal{X}} \mathrm{d}x \, h(x) \frac{f(x)}{h(x)} \\ &= \lim_{m\to\infty} \frac{1}{m} \sum_{i=1}^{m} \frac{f\left(x^{(i)}\right)}{h\left(x^{(i)}\right)} \quad \text{with probability 1.} \end{aligned}$$

Denote by $\underline{\hat{F}}$ the random variable with value given by the sample average for $\mathcal{D}$:

$$\hat{F} = \frac{1}{m} \sum_{i=1}^{m} \frac{f\left(x^{(i)}\right)}{h\left(x^{(i)}\right)}.$$

We use $\underline{\hat{F}}$ as our statistical estimator for $\underline{F}$, as we broadly described in the introductory section. Assuming a quadratic loss function, $L(\hat{F}, F) = (F - \hat{F})^2$, the bias-variance decomposition described in (2) applies *exactly*. It can be shown that the estimator $\underline{\hat{F}}$ is unbiased, that is, $\mathbb{E}(\underline{\hat{F}}) = F$, where the mean is over samples of h. Consequently, the MSE of this estimator is just its variance. The choice of sampling distribution h that minimizes this variance is given by (see Robert and Casella 2004)

$$h^{\star}(x) = \frac{|f(x)|}{\int_{\mathcal{X}} |f(x')| \, \mathrm{d}x'}.$$

By itself, this result is not very helpful, since the equation for the optimal importance distribution contains a similar integral to the one we are trying to estimate. For nonnegative integrands $f(x)$, the VEGAS algorithm (Lepage 1978) describes an adaptive method to find successively better importance distributions, by iteratively estimating $\underline{F}$ and then using that estimate to generate the next importance distribution h. In the case of these unbiased estimators, there is no trade-off between bias and variance, and minimizing MSE is achieved by minimizing variance.

Monte Carlo Optimization

Instead of a *fixed* integral to evaluate, consider a parametrized integral

$$F(\theta) = \int_{\mathcal{X}} \mathrm{d}x \; f_{\theta}(x).$$

Further, suppose we are interested in finding the value of the parameter $\theta \in \Theta$ that minimizes $F(\theta)$:

$$\theta^{\star} = \arg\min_{\theta\in\Theta} F(\theta).$$

In the case where the functional form of f_{θ} is not explicitly known, one approach to solve this problem is a technique called MCO (see Ermoliev and Norkin 1998), involving repeated MC estimation of the integral in question with adaptive modification of the parameter θ.

We proceed by analogy to the case with MC. First, we introduce the θ-indexed random variable $\underline{F}(\theta)$, all of whose components have delta-function distributions about the associated values $F(\theta)$. Next, we introduce a θ-indexed vector random variable $\underline{\hat{F}}$ with values

$$\hat{F} \equiv \left\{\hat{F}(\theta) \; \forall\, \theta \in \Theta\right\}. \qquad (4)$$

Each real-valued component $\underline{\hat{F}}(\theta)$ can be sampled and viewed as an estimate of $\underline{F}(\theta)$.

For example, let $\mathcal{D}$ be a data set as described in (3). Then for every θ, any sample of $\mathcal{D}$ provides an associated estimate

$$\hat{F}(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{f_{\theta}\left(x^{(i)}\right)}{h\left(x^{(i)}\right)}.$$

That average serves as an estimate of $\underline{F}(\theta)$. Formally, $\underline{\hat{F}}$ is a function of the random variable $\mathcal{D}$ and is given by such averaging over the elements of $\mathcal{D}$. So, a sample of $\mathcal{D}$ provides a sample of $\underline{\hat{F}}$. A priori, we make no restrictions on $\underline{\hat{F}}$, and so, in general, its components may be statistically coupled with one another. Note that this coupling arises even though we are, for simplicity, treating each function $\underline{F}(\theta)$ as having a delta-function distribution rather than as having a nonzero variance that would reflect our lack of knowledge of the $f(\theta)$ functions.

However $\underline{\hat{F}}$ is defined, given a sample of $\hat{F}$, one way to estimate $\theta^\star$ is

$$\hat{\theta}^\star = \arg\min_{\theta\in\Theta} \hat{F}(\theta).$$

We call this approach "natural" MCO. As an example, say that $\mathcal{D}$ is a set of m samples of h, and let

$$\hat{F}(\theta) \triangleq \frac{1}{m}\sum_{i=1}^{m} \frac{f_\theta\left(x^{(i)}\right)}{h\left(x^{(i)}\right)},$$

as above. Under this choice for $\underline{\hat{F}}$,

$$\hat{\theta}^\star = \arg\min_{\theta\in\Theta} \frac{1}{m}\sum_{i=1}^{m} \frac{f_\theta\left(x^{(i)}\right)}{h\left(x^{(i)}\right)}. \qquad (5)$$

We call this approach "naive" MCO.

Consider *any* algorithm that estimates $\theta^\star$ as a single-valued function of $\hat{F}$. The estimate of $\theta^\star$ produced by that algorithm is itself a random variable, since it is a function of the random variable $\underline{\hat{F}}$. Call this random variable $\underline{\hat{\theta}}^\star$, taking on values $\hat{\theta}^\star$. Any MCO algorithm is defined by $\underline{\hat{\theta}}^\star$; that random variable encapsulates the output estimate made by the algorithm.

To analyze the error of such an algorithm, consider the associated random variable given by the true parametrized integral $F(\underline{\hat{\theta}}^\star)$. The difference between a sample of $F(\underline{\hat{\theta}}^\star)$ and the true minimal value of the integral, $F(\theta^\star) = \min_\theta F(\theta)$, is the error introduced by our estimating that optimal θ as a sample of $\underline{\hat{\theta}}^\star$. Since our aim in MCO is to minimize $F(\theta)$, we adopt the loss function $L(\underline{\hat{\theta}}^\star, \theta^\star) \triangleq F(\underline{\hat{\theta}}^\star) - F(\theta^\star)$. This is in contrast to our discussion on MC integration, which involved quadratic loss. The current loss function just equals $F(\underline{\hat{\theta}}^\star)$ up to an additive constant $F(\theta^\star)$ that is fixed by the MCO problem at hand and is beyond our control. Up to that additive constant, the associated expected loss is

$$\mathbb{E}(L) = \int d\hat{\theta}^\star p(\hat{\theta}^\star) F(\hat{\theta}^\star). \qquad (6)$$

Now change coordinates in this integral from the values of the scalar random variable $\underline{\hat{\theta}}^\star$ to the values of the underlying vector random variable $\underline{\hat{F}}$. The expected loss now becomes

$$\mathbb{E}(L) = \int \mathrm{d}\hat{F}\; p(\hat{F}) F(\hat{\theta}^\star(\hat{F})).$$

The natural MCO algorithm provides some insight into these results. For that algorithm,

$$\begin{aligned}\mathbb{E}(L) &= \int \mathrm{d}\hat{F}\;\, p(\hat{F}) F\left(\arg\min_\theta \hat{F}(\theta)\right)\\ &= \int \mathrm{d}\hat{F}(\theta_1)\, \mathrm{d}\hat{F}(\theta_2)\ldots p(\hat{F}(\theta_1), \hat{F}(\theta_2), \ldots)\\ &\quad F\left(\arg\min_\theta \hat{F}(\theta)\right). \end{aligned} \qquad (7)$$

For any fixed θ, there is an error between samples of $\hat{F}(\theta)$ and the true value $F(\theta)$. Bias-variance considerations apply to this error, exactly as in the discussion of MC above. We are not, however, concerned with $\hat{F}$ for a single component θ but rather for a set Θ of θs.

The simplest such case is where the components of $\hat{F}(\Theta)$ are independent. Even so, $\arg\min_\theta \hat{F}(\theta)$ is distributed according to the laws for extrema of multiple independent random variables, and this distribution depends on higher-order moments of each random variable $\hat{F}(\theta)$. This means that $\mathbb{E}[L]$ also depends on such higher-order moments. Only the first two moments, however, arise in the bias and variance for any single θ. Thus, even in the simplest possible case, the bias-variance considerations for the individual θ do not provide a complete analysis.

In most cases, the components of $\hat{F}$ are *not* independent. Therefore, in order to analyze $\mathbb{E}[L]$, in addition to higher moments of the distribution for each θ, we must now also consider higher-order moments coupling the estimates $\hat{F}(\theta)$ for different θ.

Due to these effects, it may be quite acceptable for all the components $\hat{F}(\theta)$ to have both a large bias and a large variance, as long as they still order the θ's correctly with respect to the true $F(\theta)$. In such a situation, large covariances could ensure that if some $\hat{F}(\theta)$ were incorrectly large, then $\hat{F}(\theta'), \theta' \neq \theta$ would also be incorrectly large. This coupling between the components of $\hat{F}$ would preserve the ordering of θ's under $\underline{F}$. So, even with large bias and variance for each θ, the estimator as a whole would still work well.

Nevertheless, it *is* sufficient to design estimators $\hat{F}(\theta)$ with sufficiently small bias plus variance for each single θ. More precisely, suppose that those terms are very small on the scale of differences $F(\theta) - F(\theta')$ for any θ and θ'. Then by Chebychev's inequality, we know that the density functions of the random variables $\hat{\underline{F}}(\theta)$ and $\hat{\underline{F}}(\theta')$ have almost no overlap. Accordingly, the probability that a sample of $\hat{\underline{F}}(\theta) - \hat{\underline{F}}(\theta')$ has the opposite sign of $F(\theta) - F(\theta')$ is almost zero.

Evidently, $\mathbb{E}[L]$ is generally determined by a complicated relationship involving bias, variance, covariance, and higher moments. Natural MCO in general, and naive MCO in particular, ignore all of these effects and, consequently, often perform quite poorly in practice. In the next section, we discuss some ways of addressing this problem.

Parametric Machine Learning

There are many versions of the basic MCO problem described in the previous section. Some of the best-explored arise in parametric density estimation and parametric supervised learning, which together comprise the field of parametric machine learning (PL).

In particular, parametric supervised learning attempts to solve

$$\arg\min_{\theta \in \Theta} \int \mathrm{d}x \; p(x) \int \mathrm{d}y \; p(y \mid x) \, f_\theta(x).$$

Here, the values x represent inputs, and the values y represent corresponding outputs, generated according to some stochastic process defined by a set of conditional distributions $\{p(y \mid x),\; x \in \mathcal{X}\}$. Typically, one tries to solve this problem by casting it as an MCO problem. For instance, say we adopt a quadratic loss between a predictor $z_\theta(x)$ and the true value of y. Using MCO notation, we can express the associated supervised learning problem as finding $\arg\min_\theta F(\theta)$, where

$$\begin{aligned} l_\theta(x) &= \int \mathrm{d}y \; p(y \mid x) \, (z_\theta(x) - y)^2, \\ f_\theta(x) &= p(x) \, l_\theta(x), \\ F(\theta) &= \int \mathrm{d}x \; f_\theta(x). \end{aligned} \tag{8}$$

Next, the argmin is estimated by minimizing a sample-based estimate of the $F(\theta)$s. More precisely, we are given a "training set" of samples of $p(y \mid x)\, p(x)$, $\{(x^{(i)}, y^i)\; i = 1, \ldots, m\}$. This training set provides a set of associated estimates of $F(\theta)$:

$$\hat{F}(\theta) = \frac{1}{m} \sum_{i=1}^{m} l_\theta\left(x^{(i)}\right).$$

These are used to estimate $\arg\min_\theta F(\theta)$, exactly as in MCO. In particular, one could estimate the minimizer of $F(\theta)$ by finding the minimum of $\hat{F}(\theta)$, just as in natural MCO. As mentioned above, this MCO algorithm can perform very poorly in practice. In PL, this poor performance is called "overfitting the data."

There are several formal approaches that have been explored in PL to try to address this "overfitting the data." Interestingly, none are based on direct consideration of the random variable $F(\hat{\theta}^\star(\hat{\underline{F}}))$ and the ramifications of its distribution for expected loss (cf. (7)). In particular, no work has applied the mathematics of extrema of multiple random variables to analyze the bias-variance-covariance trade-offs encapsulated in (7).

The PL approach that perhaps comes closest to such direct consideration of the distribution of $F(\hat{\underline{\theta}}^\star)$ is uniform convergence theory, which is a central part of computational learning theory (see Angluin 1992). Uniform convergence theory starts by crudely encapsulating the quadratic loss formula for expected loss under natural MCO (7). It does this by considering the worst-case bound, over possible $p(x)$ and $p(y \mid x)$, of the probability that $F(\underline{\theta}^\star)$ exceeds $\min_\theta F(\theta)$ by more than κ. It then examines how that bound varies with κ.

In particular, it relates such variation to characteristics of the set of functions $\{f_\theta : \theta \in \Theta\}$, e.g., the "VC dimension" of that set (see Vapnik 1982, 1995).

Another, historically earlier approach, is to apply bias-plus-variance considerations to the *entire* PL algorithm $\underline{\hat{\theta}}^\star$ rather than to each $\underline{\hat{F}}(\theta)$ separately. This approach is applicable for algorithms that do not use natural MCO and even for nonparametric supervised learning. As formulated for parametric supervised learning, this approach combines the formulas in (8) to write

$$F(\theta) = \int \mathrm{d}x\, \mathrm{d}y\;\; p(x)p(y \mid x)(z_\theta(x) - y)^2.$$

This is then substituted into (6), giving

$$\begin{aligned}\mathbb{E}[L] &= \int \mathrm{d}\hat{\theta}^\star \mathrm{d}x\, \mathrm{d}y\;\; p(x)\, p(y \mid x)\, p(\hat{\theta}^\star)\\ &\qquad (z_{\hat{\theta}^\star}(x) - y)^2\\ &= \int \mathrm{d}x\, p(x) \Bigg[\int \mathrm{d}\hat{\theta}^\star \mathrm{d}y\;\; p(x)p(y\,|\,x)p(\hat{\theta}^\star)\\ &\qquad (z_{\hat{\theta}^\star}(x) - y)^2\Bigg]. \qquad (9)\end{aligned}$$

The term in square brackets is an x-parameterized expected quadratic loss, which can be decomposed into a bias, variance, etc., in the usual way. This formulation eliminates any direct concern for issues like the distribution of extrema of multiple random variables, covariances between $\underline{\hat{F}}(\theta)$ and $\underline{\hat{F}}(\theta')$ for different values of θ, and so on.

There are numerous other approaches for addressing the problems of natural MCO that have been explored in PL. Particularly important among these are Bayesian approaches, e.g., Buntine and Weigend (1991), Berger (1985), and Mackay (2003). Based on these approaches, as well as on intuition, many powerful techniques for addressing data-overfitting have been explored in PL, including regularization, cross-validation, stacking, bagging, etc. Essentially all of these techniques can be applied to *any* MCO problem, not just PL problems. Since many of these techniques can be justified using (9), they provide a way to exploit the bias-variance tradeoff in other domains besides PL.

PLMCO

In this section, we illustrate how PL techniques that exploit the bias-variance decomposition of (9) can be used to improve an MCO algorithm used in a domain outside of PL. This MCO algorithm is a version of adaptive importance sampling, somewhat similar to the CE method (Rubinstein and Kroese 2004), and is related to function smoothing on continuous spaces. The PL techniques described are applicable to any other MCO problem, and this particular one is chosen just as an example.

MCO Problem Description

The problem is to find the θ-parameterized distribution q_θ that minimizes the associated expected value of a function $G\colon \mathbb{R}^n \to \mathbb{R}$, i.e., find

$$\arg\min_\theta \mathbb{E}_{q_\theta}[G].$$

We are interested in versions of this problem where we do not know the functional form of G, but can obtain its value $G(x)$ at any $x \in \mathcal{X}$. Similarly we cannot assume that G is smooth, nor can we evaluate its derivatives directly. This scenario arises in many fields, including black-box optimization (see Wolpert et al. 2006) and risk minimization (see Ermoliev and Norkin 1998).

We begin by expressing this minimization problem as an MCO problem. We know that

$$\mathbb{E}_{q_\theta}[G] = \int_{\mathcal{X}} \mathrm{d}x\, q_\theta(x)G(x)$$

Using MCO terminology, $f_\theta(x) = q_\theta(x)G(x)$ and $F(\theta) = \mathbb{E}_{q_\theta}[G]$. To apply MCO, we must define a vector-valued random variable $\underline{\hat{F}}$ with components indexed by θ and then use a sample of $\underline{\hat{F}}$ to estimate $\arg\min_\theta \mathbb{E}_{q_\theta}[G]$. In particular, to apply naive MCO to estimate $\arg\min_\theta \mathbb{E}_{q_\theta}(G)$, we first i.i.d. sample a density function $h(x)$. By evaluating the associated values of $G(x)$, we get a data set

$$\begin{aligned}\mathcal{D} &\equiv (\mathcal{D}_{\mathcal{X}}, \mathcal{D}_G)\\ &= \Big(\{x^{(i)} : i = 1, \ldots, m\},\\ &\qquad \{G(x^{(i)}) : i = 1, \ldots, m\}\Big).\end{aligned}$$

The associated estimates of $F(\theta)$ for each θ are

$$\hat{F}(\theta) \triangleq \frac{1}{m}\sum_{i=1}^{m}\frac{q_\theta\left(x^{(i)}\right)G\left(x^{(i)}\right)}{h\left(x^{(i)}\right)}. \qquad (10)$$

The associated naive MCO estimate of $\arg\min_\theta \mathbb{E}_{q_\theta}[G]$ is

$$\hat{\theta}^\star \equiv \arg\min_\theta \hat{F}(\theta).$$

Suppose Θ includes all possible density functions over x's. Then the q_θ minimizing our estimate is a delta function about the $x^{(i)} \in \mathcal{D}_\mathcal{X}$ with the lowest associated value of $G(x^{(i)})/h(x^{(i)})$. This is clearly a poor estimate in general; it suffers from "data-overfitting." Proceeding as in PL, one way to address this data-overfitting is to use regularization. In particular, we can use the entropic regularizer, given by the negative of the Shannon entropy $S(q_\theta)$. So we now want to find the minimizer of $\mathbb{E}_{q_\theta}[G(x)]-TS(q_\theta)$, where T is the regularization parameter. Equivalently, we can minimize $\beta\mathbb{E}_{q_\theta}[G(x)] - S(q_\theta)$, where $\beta = 1/T$. This changes the definition of $\hat{F}$ from the function given in (10) to

$$\hat{F}(\theta) \triangleq \frac{1}{m}\sum_{i=1}^{m}\frac{\beta\, q_\theta\left(x^{(i)}\right)G\left(x^{(i)}\right)}{h\left(x^{(i)}\right)} - S(q_\theta).$$

Solution Methodology

Unfortunately, it can be difficult to find the θ globally minimizing this new $\hat{F}$ for an arbitrary $\mathcal{D}$. An alternative is to find a close approximation to that optimal θ. One way to do this is as follows. First, we find minimizer of

$$\frac{1}{m}\sum_{i=1}^{m}\frac{\beta\, p\left(x^{(i)}\right)G\left(x^{(i)}\right)}{h\left(x^{(i)}\right)} - S(p) \qquad (11)$$

over the set of all possible distributions $p(x)$ with domain $\mathcal{X}$. We then find the q_θ that has minimal Kullback–Leibler (KL) divergence from this p, evaluated over $\mathcal{D}_\mathcal{X}$. That serves as our approximation to $\arg\min_\theta \hat{F}(\theta)$ and therefore as our estimate of the θ that minimizes $\mathbb{E}_{q_\theta}(G)$.

The minimizer p of (11) can be found in closed form; over $\mathcal{D}_\mathcal{X}$, it is the Boltzmann distribution $p^\beta\left(x^{(i)}\right) \propto \exp\left(-\beta\, G\left(x^{(i)}\right)\right)$. The KL divergence in $\mathcal{D}_\mathcal{X}$ from this Boltzmann distribution to q_θ is

$$\begin{aligned} F(\theta) &= \mathrm{KL}\left(p^\beta \| q_\theta\right) \\ &= \int_\mathcal{X} \mathrm{d}x\, p^\beta(x)\log\left(\frac{p^\beta(x)}{q_\theta(x)}\right). \end{aligned}$$

The minimizer of this KL divergence is given by

$$\theta^\dagger = \arg\min_\theta - \sum_{i=1}^{m}\frac{\exp\left(-\beta G\left(x^{(i)}\right)\right)}{h\left(x^{(i)}\right)}\log\left(q_\theta\left(x^{(i)}\right)\right). \qquad (12)$$

This approach is an approximation to a regularized version of the naive MCO estimate of the θ that minimizes $\mathbb{E}_{q_\theta}(G)$. The application of the technique of regularization in this context has the same motivation as it does in PL: to reduce bias plus variance.

Log-Concave Densities

If q_θ is log-concave in its parameters θ, then the minimization problem in (12) is a convex optimization problem, and the optimal parameters can be found closed-form. Denote the likelihood ratios by $s^{(i)} = \exp(-\beta G(x^{(i)}))/h(x^{(i)})$. Differentiating (12) with respect to the parameters μ and Σ^{-1} and setting them to zero yields

$$\mu^\star = \frac{\sum_\mathcal{D} s^{(i)} x^{(i)}}{\sum_\mathcal{D} s^{(i)}}$$

$$\Sigma^\star = \frac{\sum_\mathcal{D} s^{(i)}\left(x^{(i)}-\mu^\star\right)\left(x^{(i)}-\mu^\star\right)^T}{\sum_\mathcal{D} s^{(i)}}$$

Mixture Models

The single Gaussian is a fairly restrictive class of models. Mixture models (see ▶ Mixture Modeling) can significantly improve flexibility, but at the cost of convexity of the KL distance minimization problem. However, a plethora of techniques from supervised learning, in particular the expectation maximization (EM) algorithm, can be applied with minor modifications.

Suppose q_θ is a mixture of M Gaussians, that is, $\theta = (\mu, \Sigma, \phi)$ where ϕ is the mixing p.m.f, we can view the problem as one where a hidden variable z decides which mixture component each sample is drawn from. We then have the optimization problem

$$\text{minimize} - \sum_{\mathcal{D}} \frac{p\left(x^{(i)}\right)}{h\left(x^{(i)}\right)} \log\left(q_\theta\left(x^{(i)}, z^{(i)}\right)\right).$$

Following the standard EM procedure, we get the algorithm described in (13). Since this is a nonconvex problem, one typically runs the algorithm multiple times with random initializations of the parameters.

$$\text{E-step: For each i, set} \quad Q_i\left(z^{(i)}\right) = p\left(z^{(i)}|x^{(i)}\right),$$

$$\text{that is,} \quad w_j^{(i)} = q_{\mu,\Sigma,\phi}\left(z^{(i)} = j|x^{(i)}\right), \quad j = 1, \dots, M.$$

$$\text{M-step: Set} \quad \mu_j = \frac{\sum_{\mathcal{D}} w_j^{(i)} s^{(i)} x^{(i)}}{\sum_{\mathcal{D}} w_j^{(i)} s^{(i)}},$$

$$\Sigma_j = \frac{\sum_{\mathcal{D}} w_j^{(i)} s^{(i)} \left(x^{(i)} - \mu_j\right)\left(x^{(i)} - \mu_j\right)^T}{\sum_{\mathcal{D}} w_j^{(i)} s^{(i)}},$$

$$\phi_j = \frac{\sum_{\mathcal{D}} w_j^{(i)} s^{(i)}}{\sum_{\mathcal{D}} s^{(i)}}.$$

Test Problems

To compare the performance of this algorithm with and without the use of PL techniques, we use a couple of very simple academic problems in two and four dimensions – the Rosenbrock function in two dimensions, given by

$$G_R(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2,$$

and the Woods function in four dimensions, given by given by

$$\begin{aligned} G_{\text{Woods}}(x) = {} & 100(x_2 - x_1)^2 + (1 - x_1)^2 \\ & +90(x_4 - x_3^2)^2 + (1 - x_3)^2 \\ & +10.1\left[(1 - x_2)^2 + (1 - x_4)^2\right] \\ & +19.8(1 - x_2)(1 - x_4). \end{aligned}$$

For the Rosenbrock, the optimum value of 0 is achieved at $x = (1, 1)$, and for the Woods problem, the optimum value of 0 is achieved at $x = (1, 1, 1, 1)$.

Application of PL Techniques

As mentioned above, there are many PL techniques beyond regularization that are designed to optimize the trade-off between bias and variance. So having cast the solution of $\arg\min_{q_\theta} \mathbb{E}(G)$ as an MCO problem, we can apply those other PL techniques instead of (or in addition to) entropic regularization. This should improve the performance of our MCO algorithm, for the exact same reason that using those techniques to trade off bias and variance improves performance in PL. We briefly mention some of those alternative techniques here.

The overall MCO algorithm is broadly described in Algorithm 1. For the Woods problem, 20 samples of x are drawn from the updated q_θ at each iteration, and for the Rosenbrock, 10 samples. For comparing various methods and plotting purposes, 1,000 samples of $G(x)$ are drawn to evaluate $\mathbb{E}_{q_\theta}[G(x)]$. Note: in an actual optimization, we will not be drawing these test

Algorithm 1 Overview of pq minimization using Gaussian mixtures

```
Draw uniform random samples on $X$
Initialize regularization parameter $\beta$
Compute $G(x)$ values for those samples
repeat
    Find a mixture distribution $q_\theta$ to minimize sampled $pq$ KL distance
    Sample from $q_\theta$
    Compute $G(x)$ for those samples
    Update $\beta$
until Termination
Sample final $q_\theta$ to get solution(s).
```

samples! All the performance results in Fig. 1 are based on 50 runs of the PC algorithm, randomly initialized each time. The sample mean performance across these runs is plotted along with 95 % confidence intervals for this sample mean (shaded regions).

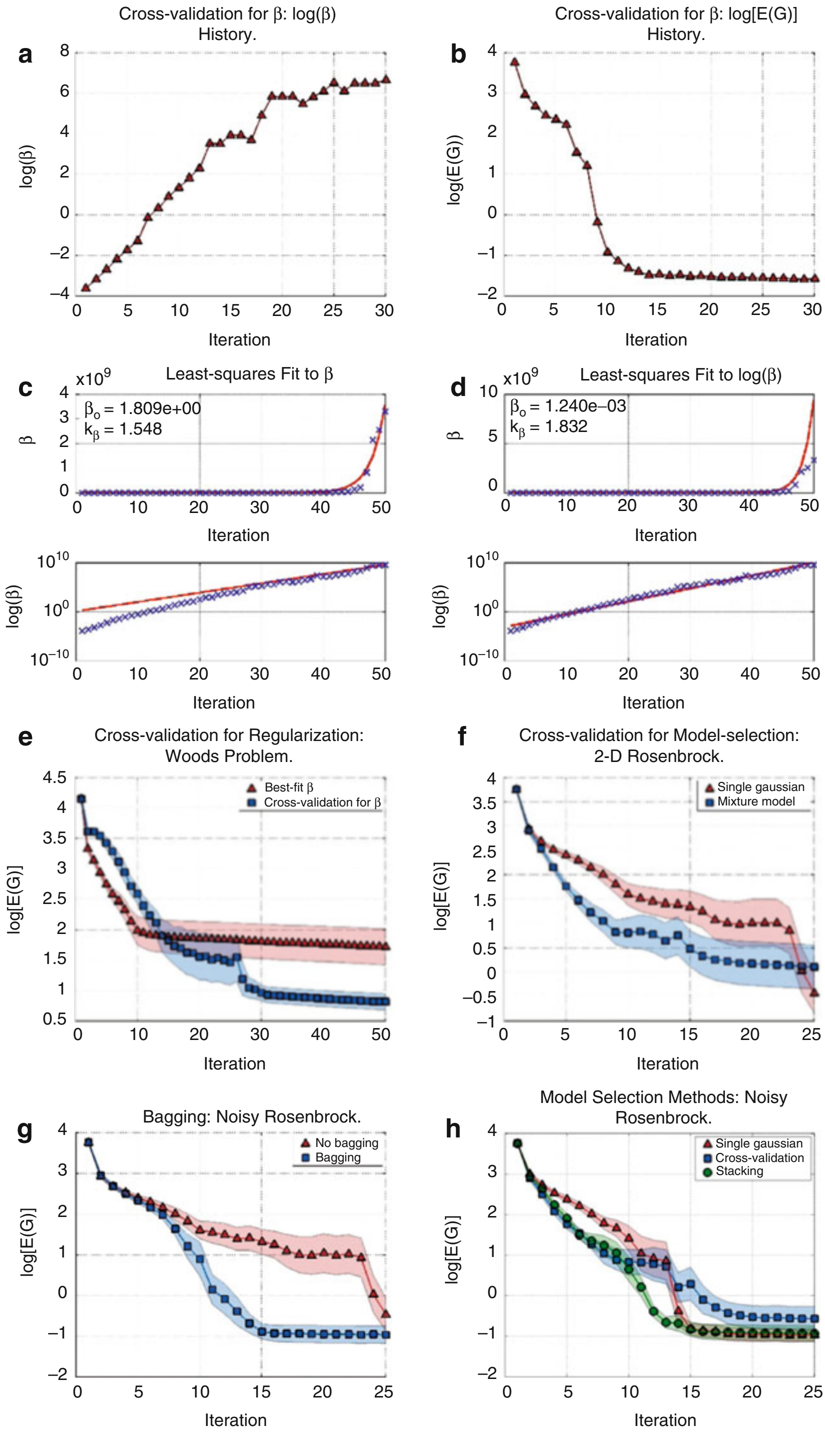
▸ *Cross-Validation for Regularization:* We note that we are using regularization to reduce variance, but that regularization introduces bias. As is done in PL, we use standard k-fold cross-validation to trade-off this bias and variance. We do this by partitioning the data into k disjoint sets. The held-out data for the ith fold is just the ith partition, and the held-in data is the union of all other partitions. First, we "train" the regularized algorithm on the held-in data $\mathcal{D}_t$ to get an optimal set of parameters $\theta^\star$ and then "test" this $\theta^\star$ by considering unregularized performance on the held-out data $\mathcal{D}_v$. In our context, "training" refers to finding optimal parameters by KL distance minimization using the held-in data, and "testing" refers to estimating $\mathbb{E}_{q_\theta}[G(x)]$ on the held-out data using the following formula (Robert and Casella 2004).

$$\widehat{g}(\theta) = \frac{\displaystyle\sum_{\mathcal{D}_v} \frac{q_\theta\left(x^{(i)}\right) G\left(x^{(i)}\right)}{h\left(x^{(i)}\right)}}{\displaystyle\sum_{\mathcal{D}_v} \frac{q_\theta\left(x^{(i)}\right)}{h\left(x^{(i)}\right)}}.$$

We do this for several values of the regularization parameter β in the interval $k_1\beta < \beta < k_2\beta$ and choose the one that yield the best held-out performance, averaged over all folds. For our experiments, $k_1 = 0.5, k_2 = 3$, and we use five equally spaced values in this interval. Having found the best regularization parameter in this range, we then use *all* the data to minimize KL distance using this optimal value of β. Note that all cross-validation is done *without* any additional evaluations of $G(x)$. Cross-validation for β in PC is similar to optimizing the annealing schedule in simulated annealing. This "auto-annealing" is seen in Fig. 1a, which shows the variation of β with iterations of the Rosenbrock problem. It can be seen that β value sometimes decreases from one iteration to the next. This can never happen in any kind of "geometric annealing schedule," $\beta \leftarrow k_\beta \beta,\ k_\beta > 1$, of the sort that is often used in most algorithms in the literature. In fact, we ran 50 trials of this algorithm on the Rosenbrock and then computed a best-fit geometric variation for β, that is, a nonlinear least squares fit to variation of β and a linear least squares fit to the variation of $\log(\beta)$. These are shown in Fig. 1c, d. As can be seen, neither is a very good fit. We then ran 50 trials of the algorithm with the fixed update rule obtained by best-fit to $\log(\beta)$ and found that the adaptive setting of β using cross-validation performed an order of magnitude better, as shown in Fig. 1e.

Cross-Validation for Model Selection: Given a set Θ (sometimes called a model class) to choose θ from, we can find an optimal $\theta \in \Theta$. But how do we choose the set Θ? In PL, this is done using cross-validation. We choose that set Θ such that $\arg\min_{\theta\in\Theta} \hat{F}(\theta)$ has the best held-out performance. As before, we use that model class Θ that yields the lowest estimate of $\mathbb{E}_{q_\theta}[G(x)]$ on the held-out data. We demonstrate the use of this PL technique for minimizing the Rosenbrock problem, which has a long curved valley that is poorly approximated by a single Gaussian. We use cross-validation to choose between a Gaussian mixture with up to four components. The improvement in performance is shown in Fig. 1d.

Bagging: In bagging (Breiman 1996a), we generate multiple data sets by resampling the given data set with replacement. These new data sets will, in general, contain replicates. We "train" the



Bias-Variance Trade-Offs: Novel Applications, Fig. 1 Various PL techniques improve MCO performance learning algorithm on each of these resampled data sets and average the results. In our case, we average the q_θ got by our KL divergence minimization on each data set. PC works even on stochastic objective functions, and on the noisy Rosenbrock, we implemented PC with bagging by resampling ten times and obtained significant performance gains, as seen in Fig. 1g.

Stacking: In bagging, we combine estimates of the same learning algorithm on different data sets generated by resampling, whereas in stacking (Breiman 1996b; Smyth and Wolpert 1999), we combine estimates of different learning algorithms on the same data set. These combined estimated are often better than any of the single estimates. In our case, we combine the q_θ obtained from our KL divergence minimization algorithm using multiple models Θ. Again, Fig. 1h shows that cross-validation for model selection performs better than a single model, and stacking performs slightly better than cross-validation.

Conclusions

The conventional goal of reducing bias plus variance has interesting applications in a variety of fields. In straightforward applications, the bias-variance trade-offs can decrease the MSE of estimators, reduce the generalization error of learning algorithms, and so on. In this article, we described a novel application of bias-variance trade-offs: we placed bias-variance trade-offs in the context of MCO and discussed the need for higher moments in the trade-off, such as a bias-variance-covariance trade-off. We also showed a way of applying just a bias-variance trade-off, as used in parametric learning, to improve the performance of MCO algorithms.

Recommended Reading

Angluin D (1992) Computational learning theory: survey and selected bibliography. In: Proceedings of the twenty-fourth annual ACM symposium on theory of computing, Victoria. ACM, New York

Berger JO (1985) Statistical decision theory and bayesian analysis. Springer, New York

Breiman L (1996a) Bagging predictors. Mach Learn 24(2):123–140

Breiman L (1996b) Stacked regression. Mach Learn 24(1):49–64

Buntine W, Weigend A (1991) Bayesian backpropagation. Complex Syst 5:603–643

Ermoliev YM, Norkin VI (1998) Monte carlo optimization and path dependent nonstationary laws of large numbers. Technical Report IR-98-009. International Institute for Applied Systems Analysis, Austria

Lepage GP (1978) A new algorithm for adaptive multidimensional integration. J Comput Phys 27:192–203

Mackay D (2003) Information theory, inference, and learning algorithms. Cambridge University Press, Cambridge

Robert CP, Casella G (2004) Monte Carlo statistical methods. Springer, New York

Rubinstein R, Kroese D (2004) The cross-entropy method. Springer, New York

Smyth P, Wolpert D (1999) Linearly combining density estimators via stacking. Mach Learn 36(1–2):59–83

Vapnik VN (1982) Estimation of dependences based on empirical data. Springer, New York

Vapnik VN (1995) The nature of statistical learning theory. Springer, New York

Wolpert DH (1997) On bias plus variance. Neural Comput 9:1211–1244

Wolpert DH, Rajnarayan D (2007) Parametric learning and monte carlo optimization. arXiv:0704.1274v1 [cs.LG]

Wolpert DH, Strauss CEM, Rajnarayan D (2006) Advances in distributed optimization using probability collectives. Adv Complex Syst 9(4):383–436

Bias-Variance-Covariance Decomposition

The bias-variance-covariance decomposition is a theoretical result underlying ▶ ensemble learning algorithms. It is an extension of the ▶ bias-variance decomposition, for linear combinations of models. The expected squared error of the ensemble $\bar{f}(x)$ from a target d is:

$$\mathcal{E}_{\mathcal{D}}\{\bar{f}(x) - d)^2\} = \overline{\text{bias}}^2 + \frac{1}{T}\overline{\text{var}} + \left(1 - \frac{1}{T}\right)\overline{\text{covar}}.$$

The error is composed of the average bias of the models, plus a term involving their average variance, and a final term involving their average *pairwise covariance*. This shows that while a single model has a two-way bias-variance tradeoff, an ensemble is controlled by a three-way tradeoff. This ensemble tradeoff is often referred to as the *accuracy-diversity* dilemma for an ensemble. See ▶ ensemble learning for more details.

Bilingual Lexicon Extraction

Bilingual lexicon extraction is the task of automatically identifying a terms in a first language and terms in a second language which are translation f one another. In this context, a term can be either a single word or an expression composed of several words the full meaning of which cannot be derived compositionally from the meaning of the individual words. Bilingual lexicon extraction is itself a form of ▶ cross-lingual text mining and is an essential preliminary step in many approaches for performing other ▶ cross-lingual text mining tasks.

Binning

▶ Discretization

Biological Learning: Synaptic Plasticity, Hebb Rule and Spike Timing Dependent Plasticity

Wulfram Gerstner
Brain Mind Institute, Lausanne EPFL, Lausanne, Switzerland

Synonyms

Correlation-based learning; Hebb rule; Hebbian learning

Definition

The brain of humans and animals consists of a large number interconnected neurons. Learning in biological neural systems is thought to take place by changes in the connections between these neurons. Since the contact points between two neurons are called synapses, the change in the connection strength is called synaptic plasticity. The mathematical description of synaptic plasticity is called a (biological) learning rule. Most of these biological learning rules can be categorized in the context of machine learning as unsupervised learning rules, and the remaining ones as reward-based or reinforcement learning. The Hebb rule is an example of an unsupervised correlation-based learning rule formulated on the level of neuronal firing rates. Spike-timing-dependent plasticity (STDP) is an unsupervised learning rule formulated on the level of spikes. Modulation of learning rates in a Hebb rule or STDP rule by a diffusive signal carrying reward-related information yields a biologically plausible form of a reinforcement learning rule.

Motivation and Background

Humans and animals can adapt to environmental conditions and learn new tasks. Learning becomes measurable by changes in the behavior: humans and animals get better at seeing and distinguishing visual objects with experience; animals can learn to go to a target location; humans can memorize a list words and recall the items 2 days later. How learning is implemented in the biological substrate is only partially known.

The brain consists billions of neurons. Each neuron has long wire-like extensions and makes contacts with thousands of other neurons. This network of neurons is not fixed but constantly changes. Connections can be formed or can disappear, and existing connections can be strengthened or weakened. Neuroscientists have shown in numerous experiments that changes can be induced by stimulating neuronal activity in an appropriate fashion. Moreover, changes in synaptic connections that have been induced in one or a few seconds can persist for hours or days, an effect called long-term potentiation (LTP) or long-term depression (LTD) of synapses.

The question arises of whether such long-lasting changes in connections are useful for learning. To answer this question, research in theoretical and computational neuroscience needs to solve two problems: First, develop a compact but realistic description of the phenomenon of synaptic plasticity observed in biology, i.e., extract learning rules from the biological data; and second, study the functional consequences of these learning rules. An important insight from experiments on LTP is that the activation of a synaptic connection alone does not lead to a long-lasting change; however, if the activation of the synapses by presynaptic signals is combined with some activation of the postsynaptic neuron, then a long-lasting change of the synapse may occur. The coactivation of presynaptic and postsynaptic neurons as a condition for learning is the key ingredient of Hebbian learning rules. Here, activation of the presynaptic neuron means that it fires one or several action potentials; activation of the postsynaptic neuron can be represented by high firing rates, a few well-timed action potentials or input from other neurons that lead to an increase in the membrane voltage.

Structure of the Learning System

The Hebb Rule

Hebbian learning rules are local, i.e., they depend only on the state of the presynaptic and postsynaptic neurons plus possibly the current value of the synaptic weight itself. Let w_{ij} denotes the weight between a presynaptic neuron j and a postsynaptic neuron i, and let us describe the activity (e.g., the firing rate) each neuron by a continuous variable ν_j and ν_i, respectively. Mathematically, we may therefore write for a local learning rule

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = F(w_{ij}; \nu_i, \nu_j) \tag{1}$$

where F is an unknown function. In addition to locality, Hebbian learning requires some kind of cooperation or correlation between the activity of the presynaptic neuron and that of the postsynaptic neuron. At the moment we restrict ourselves to the requirement of *simultaneous activity* of presynaptic and postsynaptic neurons. Since F is a function of the rates ν_i and ν_j, we may expand F about $\nu_i = \nu_j = 0$. An expansion to second order of the rates yields

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) \approx\, & c_0(w_{ij}) + c_1^{\text{pre}}(w_{ij})\nu_j + c_1^{\text{post}}(w_{ij})\nu_i \\ & + c_2^{\text{corr}}(w_{ij})\nu_i\nu_j + c_2^{\text{post}}(w_{ij})\nu_i^2 \\ & + c_2^{\text{pre}}(w_{ij})\nu_j^2 + O(\nu^3).\end{aligned} \tag{2}$$

Here, ν_i and ν_j are functions of time, i.e., $\nu_i(t)$ and $\nu_j(t)$ and so is the weight w_{ij}. The bilinear term $\nu_i(t)\nu_j(t)$ is sensitive to the instantaneous *correlations* between presynaptic and postsynaptic activities. It is this term that makes Hebbian learning a useful concept. The simplest implementation of Hebbian plasticity would be to require $c_2^{\text{corr}} > 0$ and set all other parameters in the expansion (2) to zero

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = c_2^{\text{corr}}(w_{ij})\nu_i\nu_j. \tag{3}$$

Equation (3) with fixed parameter $c_2^{\text{corr}} > 0$ is the prototype of Hebbian learning. However, since the activity variables ν_i and ν_j are always positive, such a rule will lead eventually to an increase of all weights in a network. Hence, some of the other terms (e.g., c_0 or c_1^{pre}) need to have a negative coefficient to make Hebbian learning stable. In passing we note that a learning rule with $c_2^{\text{corr}} < 0$ is usually called anti-Hebbian.

Oja's rule. A particular interesting case is a model with coefficients $c_2^{\text{corr}} > 0$ and $c_2^{\text{post}} < 0$, since it guarantees the normalization of the set of weights $w_{i1}, \ldots w_{iN}$ converging onto the same postsynaptic neuron i.

BCM rule. The Bienenstock–Cooper–Munro learning rule (also called BCM rule) with

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = a(w_{ij})\Phi(\nu_j - \vartheta)\nu_j \tag{4}$$

where Φ is some nonlinear function with $\Phi(0) = 0$ is a special case of (1). The parameter ϑ depends on the average firing rate.

Temporally asymmetric Hebbian learning. In the Taylor expansion (2) we focused on *instantaneous* correlations. More generally, we can use a Volterra expansion so as to also include temporal correlations with nonzero time lag. With the additional assumptions that changes are instantaneous, a Volterra expansion generates terms of the form

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} \propto \int_0^\infty [W_+(s) v_i(t) v_j(t-s) + W_-(s) v_j(t) v_i(t-s)] \mathrm{d}s \qquad (5)$$

with some functions W_+ and W_-. For reasons of causality, W_+ and W_- must vanish for $s < 0$. Since $W_+(s) \neq W_-(s)$, learning is asymmetric in time so that learning rules of the form (5) are called temporally asymmetric Hebbian learning. In the special case $W_+(s) = -W_-(s)$, we have antisymmetric Hebbian learning. The functions W_+ and W_- may depend on the present weight value.

STDP rule. STDP is a form of Hebbian learning with increased temporal resolution. In contrast to rate-based Hebb models, neuronal activity is described by the firing times of the neuron, i.e., the moments when the presynaptic and postsynaptic neurons emit action potentials. Let t_j^f denote the fth spike of the presynaptic neuron j and t_i^n the nth spike of the postsynaptic neuron i. The weight change in an STDP rule depends on the exact timing presynaptic and postsynaptic spikes

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = \sum_n \sum_f [A(w_{ij}; t - t_j^f)\delta(t - t_i^n) + B(w_{ij}; t - t_i^f)\delta(t - t_j^f)] \qquad (6)$$

where $A(x)$ and $B(x)$ are some real-valued functions with $A(w_{ij}, x) = B(w_{ij}, x) = 0$ for $x < 0$. Thus, at the moment of a postsynaptic spike the synaptic weight is updated by an amount that depends on the time $t_i^f - t_j^f$ since a previous presynaptic spike t_j^f. Similarly, at the moment of a presynaptic spike the synaptic weight is updated by an amount that depends on the time $t_j^f - t_i^f$ since a previous postsynaptic spike t_i^f. The dependence on the present value w_{ij} can be used to keep the weight in a desired range $0 < w_{ij} < w^{\max}$. A standard choice for the functions A and B is $A(w_{ij}); t - t_j^f = A_+(w_{ij}) \exp[-(t - t_j^f)/\tau_+]$ for $t - t_j^f > 0$ and zero otherwise. Similarly, $B(w_{ij}; t - t_i^n) = B_-(w_{ij}) \exp[-(t - t_i^n)/\tau_-]$ for $t - t_i^f > 0$ and zero otherwise. Here, τ_+ and τ_- are time constants in the range of 10–50 ms. The case $A_+(x) = (w^{\max} - x)c_+$ and $B_x(x) = -c_- x$ is called soft bounds. The choice $A_+(x) = c_+ \Theta(w^{\max} - x)$ and $B_x = -c_- \Theta(x)$ is called hard bounds. Here, c_+ and c–are positive constants. The term proportional to A_+ causes potentiation (weight increase), the one proportional to A_- causes depression (weight decrease) of synapses. Note that the STDP rule (6) can be interpreted as a spike-based form of temporally asymmetric Hebbian learning.

Functional Consequences of Hebbian Learning

Sensitivity to correlations. All Hebbian learning rules are sensitive to the correlations between the activity of the presynaptic neuron j and that of the postsynaptic neuron i. If the activity of the postsynaptic neuron is given by a linear sum of all inputs rates, i.e., $v_i = \gamma \Sigma_j w_{ij} v_j$, then correlations between presynaptic and postsynaptic activities can be traced back to correlations in the input. A particular clear example of learning driven by correlations in the input is Oja's learning rule applied to a statistical ensemble of inputs with zero mean. In this case, the postsynaptic neuron becomes sensitive to the dominant principal component of the input ensemble. If the neuron model is nonlinear, Hebbian learning extracts the independent components of the statistical input ensemble. These two examples show that learning by a Hebbian learning rule makes neurons adapt to the statistics of the input. While the condition of zero-mean input is biologically not realistic (because neuronal firing rates are always positive), this condition can be relaxed so that the same result is also applicable to biologically plausible learning rules.

Receptive fields and cortical maps. Neurons in the primary visual cortex of cats and monkeys respond to visual stimuli in a localized region of the visual field. This small sensitive zone is called the receptive field of the neuron. Neighboring neurons normally have very similar receptive fields. The exact location and properties of the receptive field are not fixed, but can be influenced by sensory stimulation. Models of unsupervised Hebbian learning can explain the development of receptive fields and the adaptation of cortical maps to the statistics of the ensemble of stimuli.

Beyond the Hebb rule. Standard models of Hebbian learning are formulated on the level of neuronal firing rates, a graded variable characterizing neuronal activity. However, real neurons communicate by spikes, short electrical pulses or "action potentials" with a rather stereotyped time course. Experiments have shown that the changes of synaptic efficacy depend not only on the mean firing rate of action potentials but on the relative timing of presynaptic and postsynaptic spikes on the level milliseconds. This Spike-Timing Dependent Synaptic Plasticity (STDP) can be considered a temporally more precise form of Hebbian learning. The STDP rule indicated above supposes that pairs of spikes (one presynaptic and one postsynaptic action potential) within some time window cause a weight change. However, experimentally it was shown that at least three spikes are necessary (one presynaptic and two postsynaptic spikes). Moreover, the voltage of the postsynaptic neuron matters even in the absence of spikes.

In most models of Hebbian learning and STDP, the factors $c_0, c_1^{\text{pre}} \ldots$ are constant or depend only on the synaptic weight. However, in biological context the speed of learning is often gated by neuromodulators. Since some of these neuromodulators contain reward-related information, one can think of learning as a three-factor rule where weight changes depend on presynaptic activity, postsynaptic activity, and the presence of a reward-related factor. A prominent neuro-modulator linked to reward information is dopamine. Three factor learning rules fall in the class of reinforcement learning algorithms.

Cross-References

- ▶ Dimensionality Reduction
- ▶ Reinforcement Learning
- ▶ Self-Organizing Maps

Recommended Reading

Bliss T, Gardner-Medwin A (1973) Long-lasting potentation of synaptic transmission in the dendate area of unanaesthetized rabbit following stimulation of the perforant path. J Physiol 232: 357–374

Bliss T, Collingridge G, Morris R (2003) Long-term potentiation: enhancing neuroscience for 30 years – introduction. Philos Trans R Soc Lond Ser B Biol Sci 358:607–611

Cooper L, Intrator N, Blais B, Shouval HZ (2004) Theory of cortical plasticity. World Scientific, Singapore

Dayan P, Abbott LF (2001) Theoretical neuroscience. MIT Press, Cambridge, MA

Gerstner W, Kistler WK (2002) Spiking neuron models. Cambridge University Press, Cambridge, UK

Gerstner W, Kempter R, van Hemmen JL, Wagner H (1996) Aneuronal learning rule for sub-millisecond temporal coding. Nature 383:76–78

Hebb DO (1949) The organization of behavior. Wiley, New York

Lisman J (2003) Long-term potentiation: outstanding questions and attempted synthesis. Philos Trans R Soc Lond Ser B Biol Sci 358:829–842

Malenka RC, Nicoll RA (1999) Long-term potentiation-a decade of progress? Science 285: 1870–1874

Markram H, Lübke J, Frotscher M, Sakmann B (1997) Regulation of synaptic efficacy by coincidence of postysnaptic AP and EPSP. Science 275:213–215

Schultz W, Dayan P, Montague R (1997) A neural substrate for prediction and reward. Science 275:1593–1599

Biomedical Informatics

C. David Page[1] and Sriraam Natarajan[2,3]
[1]Department of Biostatistics and Medical Informatics, University of Wisconsin Medical School, Madison, WI, USA
[2]Department of Computer Science, University of Wisconsin Medical School, Madison, WI, USA
[3]School of Informatics and Computing, Indiana University, Bloomington, IN, USA

Introduction

Recent years have witnessed a tremendous increase in the use of machine learning for biomedical applications. This surge in interest has several causes. One is the successful application of machine learning technologies in other fields such as web search, speech and handwriting recognition, agent design, spatial modeling, etc. Another is the development of technologies that enable the production of large amounts of data in the time it used to take to generate a single data point (run a single experiment). A third most recent development is the advent of electronic medical/health records (EMRs/EHRs). The drastic increase in the amount of data generated has led the biologists and clinical researchers to adopt algorithms that can construct predictive models from large amounts of data. Naturally, machine learning is emerging as a tool of choice.

In this entry, we will present a few data types and tasks involving such large-scale biological data, where machine learning techniques have been applied. For each of these data types and tasks, we first present the required background, followed by the challenges involved in addressing the tasks. Then, we present the machine learning techniques that have been applied to these data sets. Finally and most importantly, we present the lessons learned in these tasks. We hope that these lessons will be helpful to researchers who aim to apply machine learning algorithms to biological applications and equip them with useful knowledge when they collaborate with biological scientists.

Some of the data types that we present in this work are:

- Gene expression microarrays
- SNPs and genetic data
- Mass spectrometry and other proteomic data
- High-throughput screening data for drug design
- Electronic medical records (EMR) and personalized medicine

Some of the key lessons learned from all these data types include the following: (1) We can often do surprisingly well with far more features than data points if there are many highly predictive features (e.g., predicting cancer from microarray data) and if we use methods that are robust to overfitting such as *voted decision stumps* (Hardin et al. 2004; Waddell et al. 2005) (ensemble learning and decision stumps), *naive Bayes* (Golub et al. 1999; Listgarten et al. 2004), or *linear SVMs* (Furey et al. 2000; Hardin et al. 2004). (2) Bayes net learning (Friedman 2000) often does not give us causality, but active learning and time-series data help if available (Pe'er et al. 2001; Ong et al. 2002; Tucker et al. 2005; Zou and Conzen 2005). (3) Multi-relational methods are useful for EMRs or molecular data as the data in these cases are very highly relational. (4) There are more important issues than just increasing the accuracy of the learned model on these data sets. Such issues include how data was created, its comprehensibility (physicians typically want to understand the model that has been learned), and its privacy (some data sets contain private information that cannot be posted on public websites and cannot even be downloaded off-site).

The rest of the entry is organized as follows: First, we present gene expression microarrays, followed by SNPs and other genetic data. We then present mass spectrometry (MS) and related proteomic data. Next, we present high-throughput screening data for drug design, followed by EMR data and personalized medicine. For each of these data types, we motivate the problem and survey the different machine learning solutions. Finally, we conclude by outlining the lessons learned from all these data types and presenting some interesting and exciting directions for future research.

Gene Expression Microarrays

This data type was presented in detail in *AI Magazine* (Molla et al. 2004), and hence we will brief it in this section. We encourage the reader to read Molla et al. (2004) for more details on this data type. Genes are contained in the DNA of an organism. The mechanism by which proteins are produced from their corresponding genes is a two-step process. The first step is the *transcription* of a gene into a messenger RNA (mRNA), and in the second step called as *translation*, a protein is built using mRNA as a blueprint.

One property that DNA and RNA have in common is that each is a chain of chemicals called as *bases*. In the case of DNA, these bases are *adenine*, *cytosine*, *guanine*, and *thymine*, commonly referred to as A, C, G, and T, respectively. RNA has the same set of four bases, except thymine; RNA has *uracil*, commonly referred as U. An important characteristic of DNA and RNA is *complementarity*, that is, each base only binds well with its complement: A with T (or U) and G with C. As a result of complementarity, a strand of either DNA or RNA has a strong affinity toward what is known as its *reverse complement*, which is a strand of either DNA or RNA that has bases exactly complementary to the original strand. Complementarity is central to the processes of replication of the DNA and transcription.

In addition, complementarity can be used to detect specific sequences of bases within strands of DNA and RNA. This is done by first synthesizing a *probe*, a piece of DNA that is the complement of a sequence that one wants to detect, and then introducing this probe to a solution containing the genetic material (DNA or RNA) to be searched. This solution of genetic material is called the *sample*. In theory, the probe will bind to the sample if and only if the probe finds its complement in the sample (in reality, this process is often imperfect). The act of binding between a sample and probe is called *hybridization*. Prior to the experiment, a biologist labels the probe using a florescent flag. After the hybridization experiment, one can easily scan to see if the probe has hybridized to its reverse complement in the sample. This allows the molecular biologist to determine the presence or absence of the sequence in the sample.

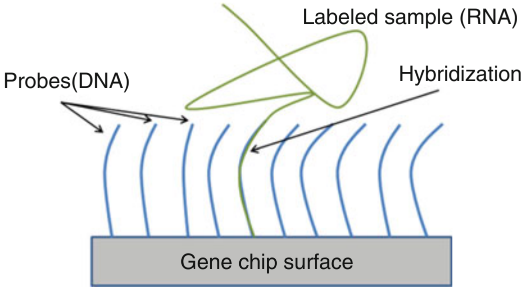
Gene Chips

DNA probe technology has been adapted for detection of tens of thousands of sequences simultaneously. This has become possible due to the device called a *microarray* or *gene chip*, the working of which is illustrated in Fig. 1. When using the chips, it is more common to label (luminescently) the samples than the probe. Thousands of copies of this labeled sample are spread across the probe, followed by washing away any copies that do not remain bound. Since the probes are attached at specific locations on the chip, if a labeled sample is detected at any position in the chip, the probe that is hybridized to its complement can be easily determined. The most common use of these gene chips is to measure the expression levels of various genes in the organism.

Probes are typically on the order of 25 bases long, whereas samples are usually about 10 times as long, with a large variation due to the process that breaks up long sequences of RNA into small samples (Molla et al. 2004).

To understand about the biology of an organism, say to understand human biology to design new drugs or lower the blood pressure or to cure diabetes, there is a necessity to understand the degree to which different genes get expressed as proteins under different conditions and different cell types. It is much easier to estimate the amount of mRNA for a gene than the protein-production rate. Microarrays provide the measurement of RNAs corresponding to the given gene rather than the amounts of protein.

In brief, experiments with the microarrays are performed as follows: As can be seen from the figure, probes are DNA strands attached to the gene chip surface. A typical probe length is



Biomedical Informatics, Fig. 1 Hybridization of sample to probe

25 bases (i.e., 25 letters from A, C, G, T to represent a gene). There may be several different subsequences of these 25 bases. Then the mRNA (which is the labeled sample) is passed over the microarrays, and the mRNA will bind to the complementary DNA corresponding to the gene better than the other DNA strings. Then the florescence levels of the different gene chips segments are measured, which in turn measures the amount of mRNA on that surface. This mRNA measurement serves as a surrogate to the expression level of the gene.

Machine Learning for Microarrays

The data from microarrays (gene chips) have been analyzed and used by machine learning researchers in two different ways:

1. Data points are genes. This is the case where the examples are genes while the features are the samples (measured expression levels of a single gene under a variety of conditions). The goal of this view is to categorize new genes based on the current set of examples.
2. Data points are samples (e.g., patients). This is the case where the examples are patients and the features are the measured expression levels of genes under one condition.

The problems have been approached in two different ways. In the unsupervised learning approach, the goal is to cluster the genes according to their expression levels or to cluster the patients (samples) based on their gene expression levels or both. Hierarchical clustering is especially widely applied. As one of many examples, see Perou et al. (1999). In the supervised learning setting, the class labels are the category of the genes or the samples. The latter is the more common supervised task, each sample being mRNA from a different patient (with the same cell type from each patient) or an organism under different conditions to learn a model that accurately predicts the class based on the features. The features could be the patient's expression values for each gene, while the class labels might be the patient's disease state. We discuss this task further in the subsequent paragraphs.

Yet another widely studied supervised learning task is to predict cancer vs. normal for a wide variety of cancer types. One of the significant lessons learned is that it is easy to predict cancer vs. normal in patients based on the gene expression by several machine learning techniques, largely regardless of the type of cancer. The main reason for this is that if cancer is present, many genes in the cancer cells "go haywire" and hence are very predictive of the cancer. The primary challenge in this prediction problem is the noise in the data (impure RNA, cross-hybridization, etc.).

Other related tasks that have been addressed include distinguishing related cancer types and distinguishing cancer from a related benign condition. An early success was a work by Golub et al. (1999), distinguishing acute myeloid leukemia and acute lymphoblastic leukemia (ALL). They used a weighted voting algorithm similar to naive Bayes and achieved a very high accuracy. This result has been repeated on this data with many other machine learning (ML) approaches. Other work examined multiple myeloma vs. benign condition. This task is challenging because the benign condition is very similar to the cancer, and hence the machine learning algorithms had a difficult time predicting accurately. We refer to Hardin et al. (2004) for more details on the experiments.

Another important lesson for machine learning researchers from this data type is that the biologists often do not want one predictive model, but a rank-ordered list of genes that a biologist can explore further with additional lab tests on certain genes. Hence, there is a need to present a small set of highly interesting genes to perform follow-up experiments on. Toward this end, statisticians have used mutual information or a t-test to rank the genes. When using a t-test, they check if the mean expression levels are different under the two conditions (cancer vs. normal), yielding a p-value. But the issue is that when working with a large number of genes (typically in the order of 30,000), there could be some genes with lower p-value by chance. This is known as the "multiple comparisons problem." One solution is to do a Bonferroni correction (multiply p-values by the number of genes), but this can be a drastic step and may eliminate all the genes. There are other methods such as false discovery rate (Storey and Tibshirani 2003) that uses the notion of q-values. We do not go into detail of this method. But the key recommendation we make is that such a method should be used along with the supervised learning method, as the biological collaborators might be interested in the ranking of genes.

One of the most important research directions for the use of microarray data lies in the prognosis and treatment. The features are the same as those of diagnosis, but the class value becomes life expectancy for a given treatment (or a positive response vs. no response to a given treatment). The goal is to use the person's genes to make these predictions. An example of this is the breast cancer prognosis study (Van't Veer et al. 2002), where the goal is to predict good prognosis (no metastasis within 5 years of initial diagnosis) vs. poor prognosis. They used an ensemble of voting algorithms and obtained very good results. Nevertheless, an important lesson learned from this experiment and others was that when using cross-validation, there is a need to tune parameters and perform feature selection *independently* on *each fold* of the cross-validation. There can be a large number of features, and it is natural to want to reduce the size of the data set before working with it. But reducing the number of features by some measure of correlation with the class, such as information gain, using the entire data set means that on each fold of cross-validation, information has leaked from the labeled test set into the training process – labels of test cases were used to eliminate many features from the training set. Hence, selecting features by looking at the entire data set can partially negate the effect of cross-validation, sometimes yielding accuracy estimates that are more than 10 % points overly optimistic. Hence the entire training process of selecting features, tuning parameters, and learning a model must be repeated for every fold in cross-validation by looking only at the training data for that fold.

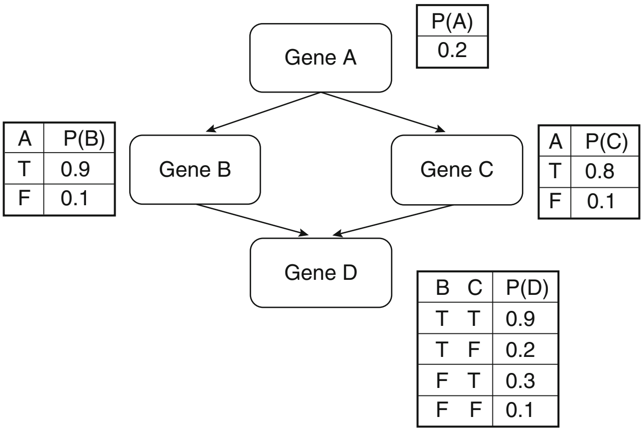
An important use of microarrays for prognosis and therapy is in the area of predictive personalized medicine (PPM). While we present the idea of PPM later in the entry, it must be mentioned that combining gene expression data with clinical trials of the patients to recommend the best treatment for the patients is a very exciting problem with promising impact in the area of PPM.

Bayesian Networks for Regulatory Pathways: Bayes nets have been one of the successful machine learning methods used for the analysis of microarray data. Recall that a Bayes net is a directed acyclic graph, such as the one shown in Fig. 2 that defines a joint distribution over the variables using a set of conditional distributions. Friedman and Halpern (1999) were the first to use Bayes nets for the microarray data type. In particular, the problem that was considered was finding regulatory pathways in genes. This problem can be posed as a supervised learning task as follows:

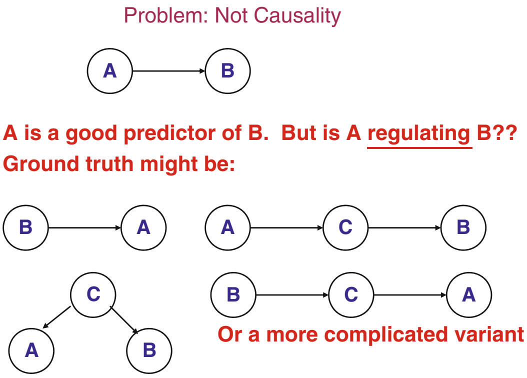
- *Given:* A set of microarray experiments for a single organism under different conditions.
- *Do:* Learn a graphical model that accurately predicts expression of some genes in terms of others.

Friedman and Halpern showed that using statistical methods, a Bayes net representing the observations (expression levels of different genes) can be learned automatically. A main advantage of Bayes nets is that they can (potentially) provide insight into the interaction networks within cells that regulate the expression of genes. But one has to exercise caution, interpreting the arcs of a learned Bayes net as representing causality. For example, in Fig. 2, one might interpret the network to mean that gene A causes gene B and gene C to be expressed, in turn influencing gene D. Note that, however, the Bayes net in this case just denotes the correlation and not the causality, that is, the direction of an arc merely represents the fact that one variable is a good predictor of the other, as illustrated in Fig. 3.

One possible method of learning causality is to use *knockout* methods (Pe'er et al. 2001), where



Biomedical Informatics, Fig. 2 A simple Bayes net. The actual learning task typically involves thousands of variables


Problem: Not Causality
A
B
A is a good predictor of B. But is A regulating B??
Ground truth might be:
B
A
A
C
B
C
A
B
B
C
A
Or a more complicated variant

Biomedical Informatics, Fig. 3 Why a learned Bayesian network may not be representing regulation of one gene by another

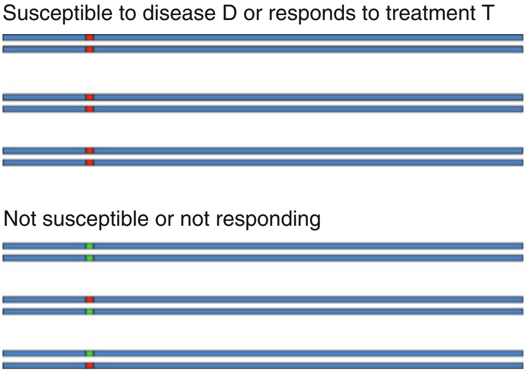
for 300 of the genes in *S. cerevisiae* (bakers' yeast), biologists have created a *knockout mutant* or a genetic mutant lacking that gene. If the parent of a gene in the Bayes net is knocked out and the child's status remains unchanged, then it is unlikely that the arc from the parent to the child captures causality. A key limitation is that the mutants are not available for many organisms. Some other approaches such as RNAi have been proposed for more efficiently doing knockouts, but a limitation is that RNAi typically reduces rather than eliminates expression of a gene.

Ong et al. (2002) used time-series data (data from the same organism at various time points) to partially address the issue of causality. They used these data to learn dynamic Bayesian networks in order to infer temporal direction for gene interactions, thereby getting a potentially better handle on causality. DBNs have been employed by other researchers for time-series gene expression data, and the approach has been extended to learn DBNs with continuous variables (Segal et al. 2005).

Single-Nucleotide Polymorphisms

Single-nucleotide polymorphisms (SNPs) are individual base positions (i.e., single-nucleotide positions) in DNA, where people (or the organism of interest) vary. Most of the variation in human DNA is due to SNP variations. (There are other variations such as copy number, insertions, and deletions that we do not consider in this entry.) There are well over three million known SNPs in humans. Technologies such as *Illumina* or *Affymetrix whole-genome* scan can measure a million SNPs in a short time. The measurement of these variations is an order of magnitude faster, easier, and cheaper than sequencing all the genes of the person.

It is believed that in the next decade, it will be possible to obtain the entire genome sequence for an individual human for under $1,000 (Mardis 2006). If we had every human's entire sequence, it could be used to predict the susceptibility of diseases for humans or the adverse reactions to drugs for a certain subset of patients. The idea is illustrated in Fig. 4. Suppose the red dots in the figure are two copies of nucleotide *A* and the green dots denote a different nucleotide, say


Susceptible to disease D or responds to treatment T
Not susceptible or not responding

Biomedical Informatics, Fig. 4 Example application of sequencing human genes. The top half is the case where patients respond to a treatment, and the bottom is the case where three patients do not respond to the treatment

C. As can be seen from the figure, people who respond to a treatment T (top half of the figure) have two copies of A (for instance, these could be the positive examples), while the people who do not respond to the treatment have at most one copy of A (negative examples and are presented in the bottom half of the figure). Now, we can imagine modeling the sequence to predict the susceptibility to a disease or responsiveness to a treatment.

SNP data can serve as a surrogate for the above problem. SNPs allow us to detect the variations among humans. An example of SNP data is presented in Fig. 5 for the prediction of *myeloma* cancer that is common with older people (with age >70) and is very rare in younger people (age <40). This data set consists of 40 people diagnosed with myeloma at young age and 40 people who weren't diagnosed till they were 70 when the disease is more common. Most SNP positions represent a pair of nucleotides and are typically restricted in the combinations of values they may assume. For example, in the figure, SNP 1 can take values from the three possible combinations $< \ CT, CC, TT \ >$ for its two positions. The goal is to use the feature values of the different SNPs to predict the class label which could be the susceptibility. That is, the goal is to determine genetic difference between people who got the disease at a young age vs. people who did not until they were old.

There is also the possibility of two patients having the same SNP pattern in the data but not the identical DNA. Patients 1 and 2 may have CT for the SNP1 and GA for SNP2, where both SNPs are on chromosome 1. But Patient 1 has C on SNP1 in the same copy of chromosome 1 as the G in SNP2, whereas Patient 2 has C on the same copy as an A. Hence, while they have the same SNP pattern of CT and GA, they do not have identical DNA. The process of converting the data from the form in the table below to the form above is called *phasing*. From a machine learning perspective, there is a choice of either working with the *unphased* data or to use an algorithm for phasing. It turns out that phasing is very difficult and is an active research area. If there are a number of unrelated patients, phasing is very hard. Hence many machine learning researchers work mainly with unphased data. Admittedly, there is a small loss of information with the unphased data that compensates for the difficulty of phasing.

Most biologists and statisticians using SNP data perform genome-wide association studies (GWAS). The goal in this work is to find individual SNPs that are significantly associated with disease, that is, such that one of the SNP values, or alleles, raises the risk of disease. This is typically measured by "relative risk" or by "odds ratio," and significance is typically measured by statistical tests such as Wald test, Score test, or LRLR (logistic regression log likelihood, where each SNP is used individually to predict disease, and log likelihood of the predictive model is compared to guessing under the null hypothesis that the SNP is not associated). One of many examples is the use of SNPs to predict susceptibility to breast cancer (Easton et al. 2007).

The advantages of SNP data compared to microarray data are the following: (1) Because SNP analysis is typically performed on DNA from saliva or peripheral blood cells, a person's SNP pattern does not change with time or disease. If the SNPs are collected from a blood sample of a person aged 40 years, the SNP patterns are probably the same as when they were born. This gives more insight to the susceptibility of the person to many diseases. Hence, we do not see the widespread changes in SNP pattern with cancer,

Person ▼ SNP▶	1	2	3	. . .	Class
Person 1	C T	A G	T T	. . .	Old
Person 2	C C	A G	C T	. . .	Young
Person 3	T T	A A	C C	. . .	Old
Person 4	C T	G G	T T	. . .	Young
. . .	. .	. . . .	.		
. . .	. .	. . . .	.		
. . .	. .	. . . .	.		

Biomedical Informatics, Fig. 5 Example of SNP data

for example, that we see in microarray data from tumor samples. (2) It is easier to collect the samples. These can be obtained from the blood samples as against obtaining, say, the biopsy of other tissue types.

The challenges of SNP data are as follows: (1) As explained earlier, the data is unphased. Algorithms exist for phasing (haplotyping), but they are error prone and do not work well with unrelated patient samples. They require the data to consist of related individuals in order to have a dense coverage. (2) Missing values are more common than in microarray data. The good news is that the amount of missing values is decreasing substantially (down from 30–40 % a few years ago to 1–2 %). (3) The sheer volume of measurements – currently, it is possible to measure a million SNPs out of over three million SNPs in the human genome. While this provides a tremendous amount of potential information, the resulting high dimensionality causes problems for machine learning. As with gene expression microarray data, we have a multiple comparisons problem, so approaches such as Bonferroni correction or q-values from false discovery rate can again be applied. But even when a significant SNP is found, it usually only increases our accuracy at predicting disease by 2 or 3 % points, because a single SNP typically either has a small effect or small penetrance (the variation is fairly rare – one value of the SNP is strongly predominant). So GWAS are missing a major opportunity to build predictive models by combining multiple SNPs with small effects – this is an exciting opportunity for machine learning.

The supervised learning task can be defined as follows:

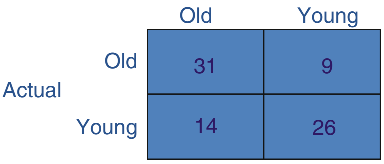
- *Given:* A set of SNP profiles each from a different patient.

Phased: Nucleotides at each SNP position on each copy of *each chromosome* constitute the *features*, and patient's *disease susceptibility* or *drug response* constitutes the *class*.

Unphased: Unordered pair of nucleotides at each SNP position constitutes the *features*, and patient's *disease susceptibility* or *drug response* constitutes the *class*.

- *Do:* Learn a model to predict the class based on the features.

We now briefly present one example of supervised learning from SNP data. Waddell et al. (2005) found that there was evidence of a genetic component in predicting the blood cancer *multiple myeloma* as it was possible to distinguish the two cases significantly better than chance (71 % accuracy). The results from using support vector machines (SVMs) are presented in Fig. 6. Similar results were obtained using a naive Bayes model as well. Listgarten et al. (2004) also used the SNP data with the goal of predicting *lung cancer*. The accuracy of 69 % obtained by them was remark-



		Old	Young
Actual	Old	31	9
	Young	14	26

Biomedical Informatics, Fig. 6 Results on predicting multiple myeloma, young (susceptible) vs. old (less susceptible), 3,000 SNPs

ably similar to the task of predicting *multiple myeloma*. The best models for predicting lung cancer were also naive Bayes and SVMs. There is a striking similarity between the two experiments on unrelated tasks using SNPs. When only the individual SNPs were considered, the accuracy for both the experiments fell to 60 %.

The lessons learned from SNP data are the following: (1) Supervised learning algorithms such as naive Bayes and SVM that can handle a large number of features in the presence of smaller number of training examples can predict disease susceptibility at rates better than chance and better than individual SNPs. (2) Accuracies are much lower than the ones with microarray data.

This is mainly due to the fact that we are predicting the susceptibility to the diseases (or the response to a drug) as against predicting whether a person already has the disease (as with the microarray data). While we are predicting using the genetic component, there are also many environmental components that are responsible for the diseases and the response. We are not considering such components in our model, and hence the accuracies are often not very high. In spite of relatively lower accuracies, they give a different valuable insight to the human gene.

We now briefly outline a couple of exciting future directions for the use of SNP data. *Pharmacogenetics* is the problem of predicting drug response from SNP profile and has been gaining momentum over the past few years. This includes predicting drug efficacy and adverse reactions to certain drugs, given a person's SNP profile. A recent New England Journal of Medicine article showed that the analysis of SNPs can significantly improve the dosing model for the most widely used orally available blood thinner, warfarin (IWPC 2009). Another exciting direction is the combination of SNP data with other data types such as clinical data that includes the history of the patient and the lab tests and microarray data. The combination of these different data sets will not only improve the accuracy of the learned model but also provide a deeper insight to the different kinds of interactions that occur within a human, such as gene interactions with other drugs.

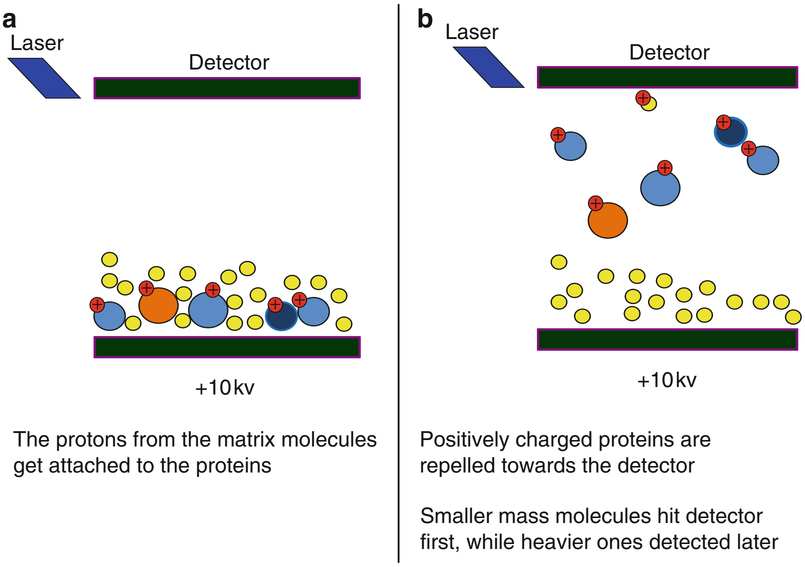
It should be mentioned that other genetic data types are becoming available and maybe useful for supervised learning as well. These data types can provide additional information about DNA sequence beyond SNPs but without the expense of full genome sequencing. They include copy-number variations and exon sequencing.

Mass Spectrometry and Proteomics

Microarrays are useful primarily because mRNA concentrations can serve as surrogates for protein concentrations and they are easier to measure. Though measuring protein concentrations directly is possible, it cannot be done in the same high-throughput manner as measuring mRNA. Recently, techniques such as *mass spectrometry* (MS or mass spec) have been successful in high-throughput measuring of proteins. Mass spec still does not give the complete coverage that microarrays provide, nor as good a quantitation.

Mass spectrometry is improving on many fronts, using many technologies. As one example, we present *time-of-flight* (*TOF*) *mass spectrometry* illustrated in Fig. 7. This measures the time required for an ionized particle starting from the sample plate (bottom of the figure) to hit the detector. The key idea is to place some proteins (indicated as larger circles) into a matrix (smaller circles are the matrix molecules). Because of mass spec limitations, the proteins typically are digested (broken into smaller peptides), for example, by the compound trypsin. When struck by a laser, the matrix molecules release protons that attach themselves to the peptides or protein fragments (shown in (a)). Note that the plate where the peptides are present

Biomedical Informatics, Fig. 7 Time-of-flight mass spectrometry

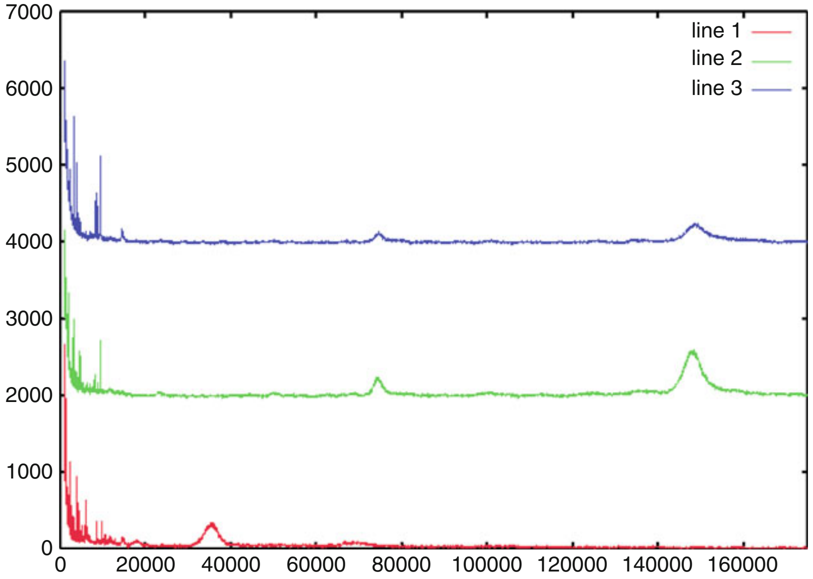

a
Laser
Detector
+10 kv
The protons from the matrix molecules get attached to the proteins
b
Laser
Detector
+10 kv
Positively charged proteins are repelled towards the detector
Smaller mass molecules hit detector first, while heavier ones detected later

is positively charged. This causes the peptides to migrate toward the detector.

As can be seen in (b) of the figure, the molecules with smaller mass move faster toward the detector. The idea is to detect the number of molecules that hit the detector at any given time. This makes it possible to use time as a surrogate for mass of the protein. The experiment is repeated a number of times, counting frequencies of "flight-times" Plotting time vs. the number of particles hitting the detector yields a spectrum as presented in Fig. 8. The figure shows three different fractions from the same sample. These kinds of spectra provide us an insight about the different types of proteins in a given sample. A technical detail is that sometimes molecules receive additional charge (additional protons) and hence fly faster. Therefore, the horizontal mass axis in a spectrum is actually a mass/charge ratio.

The main issues for machine learning researchers working with mass spectrometry data compared to microarray data are as follows: (1) There is a lot of noise in the data. The noise is due to extra peaks from handling of sample, from machine and environment (e.g., electrical noise). Also the mass to charge values may not exactly align across the spectra; the accuracy of the mass/charge values is the resolution of the mass spec. (2) Intensities (peak heights) are not calibrated across the spectra, making quantification difficult. This is to say that if one spectrum is compared to another, and if one of them has more intensity at a particular mass/charge, it does not necessarily mean that the levels of the peptide at that mass/charge are higher in that spectrum. (3) Another issue is that the mass spectrometry data is not as comprehensive as microarray data, in that it is not possible to measure all peptides (typically only several hundreds of them can be obtained). To get the best results, there is a need to fractionate the sample beforehand, getting different groups of proteins in different subsamples (fractions). (4) As already mentioned, the proteins themselves typically must be broken down (digested) into smaller peptides in order to get accurate readings from the mass spec. But this means processing is needed afterward not only to determine from a spectrum which peptides are present but also from that determination which proteins are present. It is worth noting that some of these challenges are being partially addressed by ongoing improvements in mass spectrometry technologies, including the use of "tandem mass spectrometry."

This data type opens up a lot of possibilities for machine learning research. Some of the learning tasks include:



Biomedical Informatics, Fig. 8 Example spectra from a competition by Lin et al.

- Learn to predict proteins from spectra, when the organism's proteome (full set of proteins) is known.
- Learn to identify isotopic distributions (combinations of multiple peaks for a given molecule arising from different isotypes of carbon, nitrogen, and oxygen).
- Learn to predict disease from either proteins, peaks, or isotopic distributions as features.
- Construct pathway models.

We will now present one case study that was successful and generated a lot of interest – *Early Detection of Ovarian Cancer* (Petricoin et al. 2002). Ovarian cancer is difficult to detect early, often leading to poor prognosis. The goal of this work was to predict ovarian cancer from blood samples. To this effect, the researchers trained and tested on mass spectra from blood serum. They used 100 training cases (50 positive) and used a held-out test set of 116 cases (50 positive). The results were extremely impressive (100 % sensitivity, 95 % specificity).

While the results were extremely impressive and while the machine learning methodology seemed very sound, it turns out that the preprocessing stage of the data may have introduced errors (Baggerly et al. 2004). Mass spectrometry is very sensitive to the external factors as well. For instance, if we run cancer samples on Monday and normal samples on Wednesday, it is possible that we could get differences from variations in the machine or nearby electrical equipment that is running on Monday but not Wednesday. Hence, one of the important lessons learned from this data type is the need for *careful randomization* of the data samples. This is to say that we should sample the positive and negative samples under identical conditions. It should not be the case that the positive examples are run through the machine on one day and the negatives on the other day. Any preprocessing of the data must be performed similarly.

While mass spectrometry is a widely used type of high-throughput proteomic data, other types of data are also important and are briefly covered next.

Protein Structures

X-ray crystallography and nuclear magnetic resonance are widely used to determine the three-dimensional structures of proteins. Predicting protein structures has been a very fertile field for machine learning research for several decades.

While the amino acid sequence of a protein is called its primary structure, it is more difficult to determine secondary structure and tertiary (3D) structure. Secondary structure maps subsequences of the primary structure in the three classes of alpha helix (helical structures akin to a telephone cord, often denoted by A), beta strand (which comes together with other strand sections to form planar structures called beta sheets, often denoted by B), and less descript regions referred to as coil, or loop regions, often denoted by C.

Predicting secondary structure and tertiary structure has been a popular topic for machine learning for many years, because training data exists, yet it is difficult and expensive to experimentally determine structures. We will not attempt to survey all the work in this area. Waltz and colleagues (Zhang et al. 1992) showed the benefit of applying neural networks to the task of secondary structure prediction, and the best secondary structure predictors (e.g., Rost and Sander 1993) have continued to be constructed by machine learning over the years. Approaches for predicting the tertiary structure have also relied heavily on machine learning and include ab initio prediction (e.g., Bonneau and Baker 2001), prediction aided by crystallography data (e.g., DiMaio et al. 2007), and homology-based prediction (by finding similar proteins). For over a decade, there has been a regular competition in the prediction of protein structures (Critical Assessment of Structure Prediction [CASP]).

Protein–Protein Interactions

Another proteomics data type is protein–protein interactions. This is illustrated in Fig. 9. The idea is to identify proteins that interact with the current protein say P. Generally, this is performed as follows: In the sample, there are some proteins of type X (shown in pink in the figure) and other types of proteins. Proteins that interact with X are bonded to X. Then antibodies (shown as Y-shaped green objects) are introduced in the sample. The idea of antibodies is to collect the proteins of type X. Once the antibodies have collected all protein X's in the sample, they can be analyzed through mass spectrometry presented earlier.

A particularly high-throughput way of measuring protein–protein interactions is through "ChIP-chip" data. The supervised learning tasks for this task include:

- Learn to predict protein–protein interactions: Protein three-dimensional structures may be critical.
- Use protein–protein interactions in construction of pathway models.
- Learn to predict protein function from interaction data.

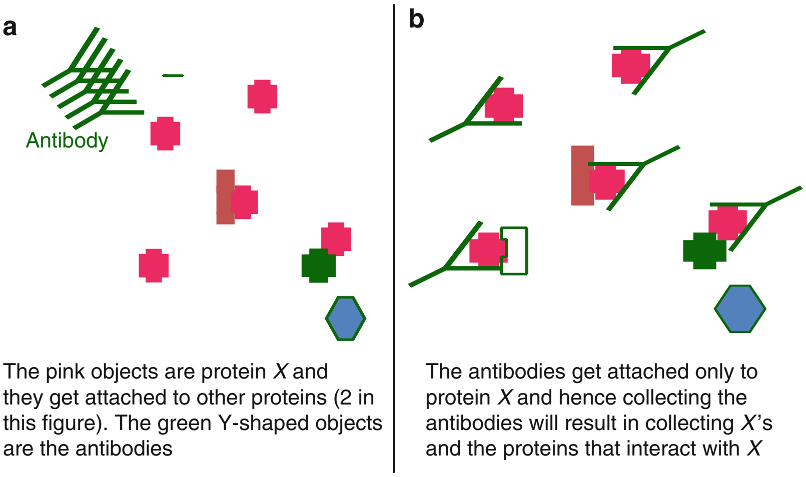
Related Data Types

- *Metabolomics* measures concentration of each low-molecular-weight molecule in sample. These typically are *metabolites*, or small molecules produced or consumed by reactions in biochemical pathways. These reactions are typically catalyzed by proteins (specifically, enzymes). This data typically uses mass spectrometry.
- *ChIP-chip* data measures protein-DNA interactions. For example, transcription factors are proteins that interact with DNA in specific locations to alter transcription of a nearby gene.
- *Lipomics* is analogous to metabolomics but measuring concentrations of lipids rather than metabolites. These potentially help induce biochemical pathway information or help in disease diagnosis or treatment choice.

High-Throughput Screening Data for Drug Design

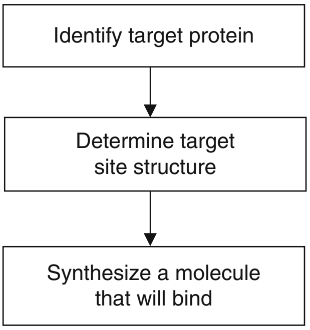
The typical steps in designing a drug are the following: (1) Identifying a target protein – for example, while developing an antibiotic, it will be useful to find a protein that belongs to the bacteria

Biomedical Informatics, Fig. 9 Schematic of antibody-based identification of protein–protein interactions



The pink objects are protein *X* and they get attached to other proteins (2 in this figure). The green Y-shaped objects are the antibodies

The antibodies get attached only to protein *X* and hence collecting the antibodies will result in collecting *X*'s and the proteins that interact with *X*


Identify target protein
Determine target site structure
Synthesize a molecule that will bind

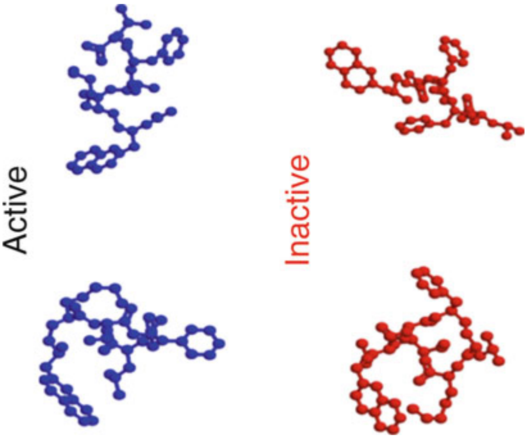
Biomedical Informatics, Fig. 10 Steps involved in drug design

that we are interested in and find a small molecule that will bind to that protein. In order to perform this, we need the knowledge of proteome/genome and the relevant biological pathways. (2) Determining the target site structure once the protein has been identified – this is typically performed using crystallography. (3) Finding a molecule that will bind to the target site. These steps are presented in Fig. 10.

The molecules that bind to the target may have a number of other problems, and hence they cannot directly be used as a drug. Some common problems are as follows: (1) They may bind too tightly or not tightly enough. (2) They may be toxic. (3) They may have unanticipated side effects in the body. (4) They may break down as soon as they get into the body or may not leave the body soon enough. (5) They may not get to the right target in the body (e.g., cross blood–brain barrier). (6) They may not diffuse from gut to bloodstream. Also, since the organisms are different, even if a molecule works in the test tube and in animal studies, it may fail in clinical trials. Also while a molecule may work for some people, it may not work for others. Conversely, while some molecules may cause harmful side effects in some people, they may not do so in others.

Often pharmaceutical companies will use robotic high-throughput screening assays to test many thousands of molecules to see if they bind to the target protein, and then computational chemists will work to determine the commonalities that allow them to bind to the target as often the structure of the target protein cannot be determined. The process of discovering the commonalities across the different molecules presents a great opportunity for machine learning research. The first study of this task using machine learning was by Dietterich, Lathrop, and Lozano-Perez and led to the formulation of multi-instance learning. Yet, another machine learning task could be to predict the reactions of the patients to the drugs.

High-Throughput Screening: When the target structure is unknown, it is a common practice to test many molecules (1,000,000) to find some that bind to the target. This is called as *high-throughput screening*. Hence, it is important to infer the shape of the target from three-



Biomedical Informatics, Fig. 11 An example of structure learning

dimensional structural similarities. The shared three-dimensional structure is called as *pharmacophore*. This is a perfect example of a machine learning task with a spatial target and is presented in Fig. 11.

Given: A set of molecules, each labeled by activity (binding affinity for a target protein) and a set of low-energy conformers for each molecule

Do: Learn a model that accurately predicts the activity (may be Boolean or real valued).

The common machine learning approaches taken toward solving this problem are:

1. Representing a molecule by thousands to millions of features and using standard techniques (KDD 2001)
2. Representing each low-energy conformer by feature vector and using multiple-instance learning (Jain et al. 1994)
3. Relational learning – using either inductive logic programming techniques (Finn et al. 1998) or graph mining

Thermolysin Inhibitors: We present some results of relational learning algorithms on thermolysin inhibitor data set (Davis 2007a). Thermolysin belongs to the family of metalloproteases and plays roles in physiological processes such as digestion and blood pressure regulation. The molecules in the data set are known inhibitors of thermolysin. Activity for these molecules is measured in $pKi = -\log Ki$, where Ki is a dissociation constant, measuring the ratio of the concentrations of bound product to unbound constituents. A higher value indicates a stronger affinity for binding. The data set that was used had the ten lowest energy conformations (as computed by the SYBYL software package [www.tripos.com]) for each of 31 thermolysin inhibitors along with their activity levels.

The key results for this data set using the relational algorithm SAYU (Davis 2007b) were:

- Ten five-point pharmacophores identified, falling into two groups (7/10 molecules):
  - Three "acceptors," one hydrophobe, and one donor
  - Four "acceptors" and one donor
- Common core of Zn ligands, Arg203, and Asn112 interactions identified
- Correct assignments of functional groups
- Correct geometry to 1 Å tolerance
- Increasing tolerance to 1.5 Å finds common six-point pharmacophore including one extra interaction

Antibacterial Peptides: This is a data set of 11 pentapeptides showing activity against *Pseudomonas aeruginosa* (Spatola et al. 1999). There are six active pharmacophores with $<64\,\mu g/ml$ of IC_{50} and five inactives. The pharmacophore that has been identified is presented in Table 1.

Dopamine Agonists: The last data set that we present here consists of dopamine agonists (Martin et al. 1993). Dopamine works as a neurotransmitter in the brain, where it plays a major role in the movement control. Dopamine agonists are molecules that function like dopamine and produce dopamine-like effects and can potentially be used to treat diseases such as Parkinson's disease. The data set had 23 dopamine agonists along with their activity levels. The pharmacophore identified using inductive logic programming is presented in Table 2.

Biomedical Informatics, Table 1 Identified pharmacophore

A molecule M is active against *Pseudomonas aeruginosa* if it has a conformation B such that:
M has a hydrophobic group C
M has a hydrogen acceptor D
The distance between C and D in conformation B is 11.7 Å
M has a positively charged atom E
The distance between C and E in conformation B is 4 Å
The distance between D and E in conformation B is 9.4 Å
M has a positively charged atom F
The distance between C and F in conformation B is 11.1 Å
The distance between D and F in conformation B is 12.6 Å
The distance between E and F in conformation B is 8.7 Å
Tolerance 1.5 Å

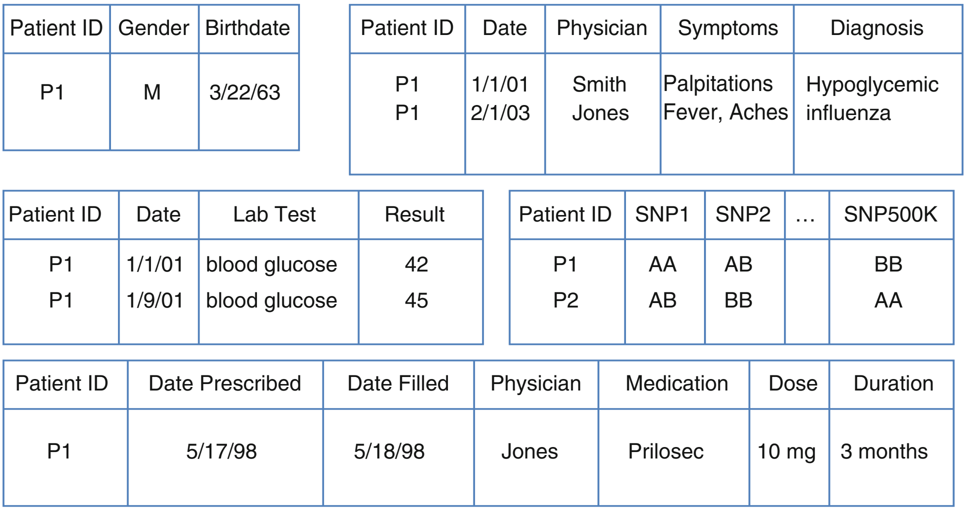
Biomedical Informatics, Table 2 Pharmacophore identified for dopamine agonists

Molecule A has the desired activity if:
• In conformation B molecule A contains a hydrogen acceptor at C • In conformation B molecule A contains a basic nitrogen group at D • The distance between C and D is 7.05966 ± 0.75 Å • In conformation B molecule A contains a hydrogen acceptor at E • The distance between C and E is 2.80871 ± 0.75 Å • The distance between D and E is 6.36846 ± 0.75 Å • In conformation B molecule A contains a hydrophobic group at F • The distance between C and F is 2.68136 ± 0.75 Å • The distance between D and F is 4.80399 ± 0.75 Å • The distance between E and F is 2.74602 ± 0.75 Å

Electronic Medical Records (EMR) and Personalized Medicine

Predictive personalized medicine (PPM) is a vision of the future, whose parts are beginning to come into place now. Under this vision, physicians can construct safer and more effective prevention and treatment plans for each patient. This is rendered possible by predicting the impact of treatments on patients – their effectiveness for different classes of patients, adverse reactions of certain drugs that are prescribed to the patients, and susceptibility of different types of patients to diseases. PPM can become a reality due to three reasons: The first is the widespread use by many clinics of *electronic medical records* (EMR, also called as *Electronic Health Records* – EHR). The second is that whole-genome scan technology makes it possible in one experiment, for well under $1,000, to measure for one patient a half million to one million SNPs or individual positions in the DNA where humans vary. The third key reason is the advancement of statistical modeling (machine learning) methods in the past decade that can handle large relational longitudinal databases with significant amount of noise. The first two reasons make it possible for the clinics to have a relational database of the form presented in Fig. 12.

Given such a database, it is conceivable to use existing machine learning algorithms for achieving the goal of PPM. These algorithms could focus on predicting which patients are at risk (positive and negative examples). Another task is predicting which patients will respond to a specific treatment – a set of patients who have undergone specific treatments in order to learn predictive models that could be extended to similar patients of the population. Similarly, it is possible to focus on certain drugs and their adverse reactions and use them to predict the adverse reactions of similar drugs that are released in the market. In this work, we focus on the machine



Patient ID	Gender	Birthdate
P1	M	3/22/63

Patient ID	Date	Physician	Symptoms	Diagnosis
P1	1/1/01	Smith	Palpitations	Hypoglycemic
P1	2/1/03	Jones	Fever, Aches	influenza

Patient ID	Date	Lab Test	Result
P1	1/1/01	blood glucose	42
P1	1/9/01	blood glucose	45

Patient ID	SNP1	SNP2	…	SNP500K
P1	AA	AB		BB
P2	AB	BB		AA

Patient ID	Date Prescribed	Date Filled	Physician	Medication	Dose	Duration
P1	5/17/98	5/18/98	Jones	Prilosec	10 mg	3 months

Biomedical Informatics, Fig. 12 Electronic health records (dramatically simplified) – most data currently do not include SNP information but are anticipated in the future

learning solutions to predict adverse drug reactions for different drugs.

There are actually at least three different tasks for machine learning in predicting adverse drug events (ADEs).

Task 1:

Given: Patient data (from claims databases and/or EMRs) and a drug D

Do: Construct a model to predict a minimum efficacious dose of drug D, because a minimum dose is less likely to induce an ADE.

An example of this task is predicting the "stable dose" of the blood thinner warfarin (Coumadin) for a patient (McCarty et al. 2005). A stable dose of Warfarin yields the desired degree of anticoagulation, whereas a higher dose can lead to bleeding ADEs; the stable dose for a patient is currently found by trial and error, modifying the dose and measuring the degree of anticoagulation. The cited study shows that a learned dosing model can predict a significantly better starting dose (significantly closer to the final "stable dose") than the 5 mg/day starting dose currently used in many clinics.

Task 2:

Given: Patient data (from claims databases and/or EMRs), a drug D, and an adverse event E

Do: Construct a model to predict which patients are likely to suffer the adverse event E if they take D.

In this second task, we assume that the association between D and E already has been hypothesized. We seek to construct models that can predict who will suffer a given event if they take the drug. Here, whether the patient will suffer adverse event E is the class variable to be predicted. This task is important for *personalized medicine*, as accurate models for this task can be used to identify patients who should not be given a particular drug. An earlier study has demonstrated the benefit of a statistical relational learning (SRL) system called SAYU (Davis 2007b) over standard machine learning approaches with a feature-vector representation of the EHR for the task of predicting which users of cox2 inhibitors would have an MI.

Task 3:

Given: Patient data (from claims databases and/or EMRs) and a drug D

Do: Determine if evidence exists that associates D with a previously unanticipated adverse event.

This third task is the most challenging because no associated event has been hypothesized. There is a need to identify the response variable to be predicted. In brief, the major approach for this task is to use machine learning "in reverse." We seek a model that can predict which patients are on drug D using the data after they start the drug (left censored) and also censoring the indications of the drug. If a model can predict (with accuracy better than chance on held-aside data) which patients are taking the drug, there must be some combination of variable settings more common among patients on the drug. Because we have left censored, in theory, this commonality should not consist of common symptoms, but common effects, presumably from the drug. The model can then be examined by the experts to see if it might indicate a possible new adverse event for the drug.

The preceding use of machine learning "in reverse" actually can be viewed as subgroup discovery (Wrobel 1997; Klösgen 2002), finding a subgroup of patients on drug D who share some subsequent clinical events. The learned model – say an if–then rule – need not correctly identify everyone on the drug but rather merely a subgroup of those on the drug, while not generating many false positives (individuals not on the drug). This task poses several different challenges that traditional ML methods will find difficult to handle.

First, the data is *multi-relational*. There are several objects such as doctors, patients, drugs, diseases, and labs that are connected through relations such as visits, prescriptions, diagnoses, etc. If traditional machine learning (ML) techniques are to be employed on this problem, they require flattening the data into a single table. All known flattening techniques such as computing a join or summary features result in either (1) changes in frequencies on which machine learning algorithms critically depend or (2) loss of information. They also typically result in loss of some correlations between the objects and explosion in database size. Second, the data is *non-i.i.d.*, as there are relationships between the objects and between different rows within a table. Third, there are *arbitrary* numbers of patient visits, diagnoses, and prescriptions for different patients. This is to say that there is no fixed pattern in the diagnoses and prescriptions of the patients. It is incorrect to assume that the patients are diagnosed a fixed number of times or to assume only the last diagnosis is relevant. To predict the adverse reactions to a drug, it is important to consider the other drugs that the patient is prescribed or has been prescribed in the past, as well as past diagnoses and laboratory results. To capture these interactions, it is critical to explicitly model time since the interactions are highly *temporal.* Some drugs taken at the same time can lead to side effects, while in some cases, drugs taken after one another cause side effects. It is important to capture such interactions to be able to make useful predictions for the physicians and the Federal Drug Authority (FDA). In this work, we focus on this hardest task and present the results on two data sets.

Cox2 Inhibitors: Recently, a study was performed to see if there were any unanticipated adverse events that occurred when subjects used cox2 inhibitors (Vioxx, Celebrex, and Bextra). Cox2 inhibitors are a nonsteroidal anti-inflammatory class of drugs that were used to reduce joint pain. Vioxx, Celebrex, and Bextra were approved for use in the late 1990s and were ranked as one of the top therapeutic drugs in the USA. Several clinical trials were conducted, and the APPROVe trial (focused on Vioxx outcomes) showed an increase of adverse events from myocardial infarction, stroke, and vascular thrombosis. The manufacturer withdrew Vioxx from the market shortly after the results were published. The other cox2 inhibitor drugs were discontinued shortly thereafter.

This study utilized the Marshfield Clinic's Personalized Medicine Research Project (McCarty et al. 2005) (PMRP) cohort consisting of approximately 19,700+ subjects. The PMRP cohort included adults aged 18 years and older, who reside in the Marshfield Epidemiology Study Area (MESA). Marshfield has one of the oldest internally developed electronic medical records (Cattails MD) in the USA, with coded diagnoses dating back to the early 1960s. Cattails MD has over 13,000 users throughout central and northern Wisconsin.

Since the data is multi-relational, an inductive logic programming (Muggleton and De Raedt 1994) system, *Aleph* (Srinivasan 2001) was used to learn the models. Aleph learns rules in the form of Prolog clauses and scores rules by positive examples covered (P) minus negative examples covered (N). Seventy-five percent of the data was used for training and rule development, while

Biomedical Informatics, Table 3 Cox2 inhibitor test data results

	Actual		
Rule	+	–	
+	438	158	596
–	269	549	818
	707	707	1,414
Accuracy	0.69801		

the remaining 25 % was used for testing. There were 14,654 subjects within the PMRP cohort that had medication records. Within this cohort, almost 20 % of the subjects indicated use of a cox2 inhibitor, and more specifically, 8.5 % indicated the use of Vioxx. Approximately, 3.5 % of this cohort had an indicated use of clopidogrel biosulfate (Plavix).

Aleph generated thousands of rules and selected a subset of the "best" rules that were based on the scoring algorithm. The authors also developed specific hypotheses to test for known adverse events to validate the approach (indicated by # A). This rule was: cox2(A):- diagnoses(A, _,'410'). It states that if finding (A), the subject would have the diagnosis coded as *410* (*myocardial infarction*). Aleph also provided summary statistics on model performance for identifying subjects on cox2 inhibitors, as indicated in Table 3. If we assume that the probability of being on the cox2 inhibitor is greater than 5 (the common threshold), then the model has a predictive probability of 69 % to predict cox2 inhibitor use.

OMOP Challenge: Observational Medical Outcomes Partnership (OMOP) designed and developed an automated procedure to construct simulated data sets to identify adverse drug events. The simulated data sets are modeled after real observational data sources but are comprised of hypothetical persons with fictional drug exposure and health outcome occurrence. The data sets are constructed such that the relationships between the fictional drugs and fictional outcomes are well characterized as *true* and *false* associations. That is, hypothetical persons are created and assigned fictional drug exposure periods and instances of health outcomes based on random sampling from probability distributions that define the relationships between the fictional drugs and outcomes. The relationships created within the simulated data sets are contrived but are representative of the types of relationships observed within real observational data sources. OMOP has made a simulated data set and the simulator itself publicly available as part of the OMOP Cup Data Mining Competition (http://omopcup.orwik.com).

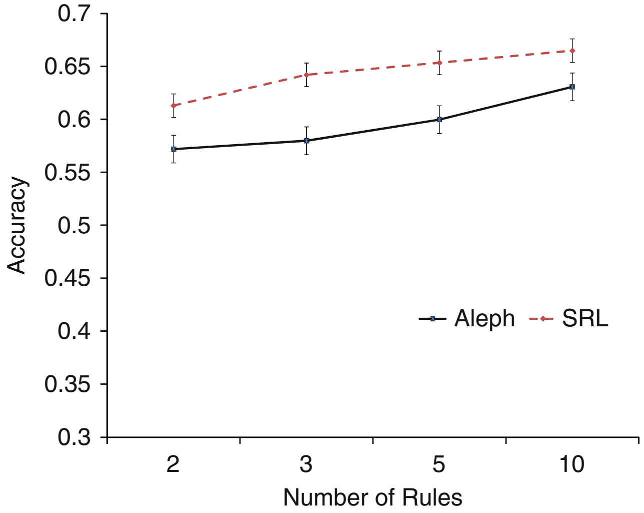
Aleph was used to learn rules from a subset of the data (about 10,000 patients). Each patient had a record of drugs and diagnoses (conditions) with dates attached. A few examples of the rules learned by Aleph in this data set are:

```
on_drug(A):-condition_occurrence(B,C,A,D, E,3450,F,G,H)

 on_drug(A):- condition_occurrence(B,C,A,D,E, 140,F,G,H)

                    condition_occurrence(I,J,A,K,L, 1487,M,N,O)
```

The first rule identifies drug *3450* as interesting, while the second rule identifies two other drugs as interesting when predicting the reaction for person A. With about 150 rules, Aleph was able to achieve a 67 % coverage. The results were compared against a statistical relational learning technique (SRL) (Getoor and Taskar 2007) that uses a probability distribution on the rules. The results are presented in Fig. 13. As expected, with a small number of rules, SRL has a better performance than Aleph, but as the number of rules increase, they converge on the same performance.

Lessons Learned: The lessons learned from the EHR data and experiments are: (1) We do not want to find patterns in the patients who get prescribed a particular drug, because we already know such patterns – they are the indications of the drug. Therefore, it is important to censor (omit) data about patients before they started the drug. Nevertheless, this left censoring is insufficient to guarantee that the learned models describe only ADEs. Some diagnoses may get repeated at a later date in the patient's record. There may be additional diagnoses, drugs, labs, or vitals that are correlated with the indication of the drug being studied. (2) Despite left censoring, a high accuracy or highly accurate discovered subgroup does not automatically mean

Biomedical Informatics, Fig. 13 Results of OMOP data



we have uncovered one or more ADEs. Instead, all rules must be vetted by a human expert to determine if they are representative of an ADE or of some other phenomenon such as that patients on arthritis medication such as cox2 inhibitors also suffer from other correlated ailments. Once these associated conditions are also censored, learning ideally should be rerun in case ADEs were masked by other rules that scored better. (3) Another lesson is that data is multi-relational, including longitudinal (temporal), and hence may be best analyzed by methods that can directly handle such data. Nevertheless, the initial approach presented earlier does not make full use of the relational nature of the data, especially of its longitudinal nature. It would be desirable to take into account time from drug exposure to events, but this is a challenging direction because different drugs can cause ADEs over different ranges of time. Some drugs may cause an ADE only within hours after they are taken, whereas others may have permanent effects that only manifest themselves as an ADE years later.

Identifying previously unanticipated ADEs, predicting who is most at risk for an ADE, and predicting safe and efficacious doses of drugs for particular patients are all important needs for society. With the recent advent of "paperless" medical record systems, the pieces are in place for machine learning to help meet these important needs.

Conclusion

In this work, we aim to survey the abundant opportunities in biomedical applications to machine learning researchers by presenting several data types to which machine learning techniques have been applied successfully or showing tremendous promise. One of the most important developments in biology and medicine over the last few years is the availability of technologies that can produce large volumes of data. This in turn has necessitated the need for processing large volumes of data in a reasonable amount of time, presenting the perfect setting for machine learning algorithms to have an impact. We outlined several data types including gene expression microarrays (measuring mRNA), mass spectrometry (measuring proteins), SNP chips (measuring genetic variation), and electronic medical/health records (EMR/EHRs).

The key lessons learned from all these data types are as follows: (1) Even if the number of features is greater than the number of data points (e.g., predicting cancer from microarray data), we can do well provided the features are highly predictive. (2) Careful randomization of data samples is necessary. (3) It is very easy to overfit the data, and hence robust techniques such as voted decision stumps, naive Bayes, or linear SVMs are in general very useful tools for such data sets. (4) Bayes nets do not give us causality, and hence knockout experiments (active learning) and DBNs with time-series data can help. (5) Multi-relational methods such as SRL and ILP are helpful for predictive personalized medicine due to the relational nature of the data. (6) Mostly, the collaborators are interested in measures other than just accuracy. Comprehensibility, privacy, and ranking are other criteria that are important to biologists.

This chapter is necessarily incomplete because so many exciting tasks and data types exist within biology and medicine. While we have touched on many of the leading such data types, other related ones also exist. For example, there are many opportunities in analyzing genomic and protein sequences (► Learning Models of Biological Sequences). Other opportunities exist within phylogenetics, for example, see work by Heckerman and colleagues on HIV (Carlson et al. 2009). New technologies such as optical mapping are constantly being developed and refined (Ananiev et al. 2008). Machine learning has great potential for developing models for computer-aided diagnosis (CAD), for example, for mammography (Burnside et al. 2009). Data types such as metabolomics and auxotrophic growth experiments raise opportunities for active learning and for automatic revision of biological network models, for example, as in the Robot Scientist projects (Jones et al. 2004; Oliver et al. 2009). Incorporation of multiple data types can further help in mapping out the regulatory entities and networks of an organism (Noto and Craven 2006). It is our hope that this article will encourage some machine learning researchers to delve deeper into these and other related opportunities.

Cross-References

► Learning Models of Biological Sequences

Recommended Reading

Ananiev GE, Goldstein S, Runnheim R, Forrest DK, Zhou S, Potamousis K, Churas CP, Bergendah V, Thomson JA, David C (2008). Schwartz1. Optical mapping discerns genome wide DNA methylation profiles. BMC Mol Biol 9. doi:10.1186/1471-2199-9-68.

Baggerly K, Morris JS, Combes KR (2004) Reproducibility of seldi-tof protein patterns in serum: comparing datasets from different experiments. Bioinformatics 20:777–785

Bonneau R, Baker D (2001) Ab initio protein structure prediction: progress and prospects. Ann Rev Biophys Biomol Struct 30:173–189

Burnside ES, Davis J, Chhatwal J, Alagoz O, Lindstrom MJ, Geller BM, Littenberg B, Kahn CE, Shaffer K, Page D (2009) Unique features of HLA-mediated hiv evolution in a Mexican cohort: a comparative study. Radiology 251: 663–672

Carlson J, Valenzuela-Ponce H, Blanco-Heredia J, Garrido-Rodriguez D, Garcia-Morales C, Heckerman D et al (2009) Unique features of HLA-mediated HIV evolution in a Mexican cohort: a comparative study. Retrovirology 6(72):39

Davis J, Santos Costa V, Ray S, Page D (2007a) An integrated approach to feature construction and model building for drug activity prediction. In: Proceedings of the 24th international conference on machine learning (ICML), Corvalis

Davis J, Ong I, Struyf J, Burnside E, Page D, Santos Costa V (2007b) Change of representation for statistical relational learning. In: Proceedings of the 20th international joint conference on artificial intelligence (IJCAI), Hyderabad

DiMaio F, Kondrashov D, Bitto E, Soni A, Bingman C, Phillips G, Shavlik J (2007) Creating protein models from electron-density maps using particle-filtering methods. Bioinformatics 23:2851–2858

Easton DF, Pooley KA, Dunning AM, Pharoah PD et al (2007) Genome-wide association study identifies novel breast cancer susceptibility loci. Nature 447:1087–1093

Finn P, Muggleton S, Page D, Srinivasan A (1998) Discovery of pharmacophores using the inductive logic programming system PROGOL. Mach Learn 30(1,2):241–270

Friedman N (2000) Being Bayesian about network structure. Mach Learn 50:95–125

Friedman N, Halpern J (1999) Modeling beliefs in dynamic systems. Part II: revision and update. J AI Res 10:117–167

Furey TS, Cristianini N, Duffy N, Bednarski BW, Schummer M, Haussler D (2000) Support vector classification and validation of cancer tissue samples using microarray expression. Bioinformatics 16(10):906–914

Getoor L, Taskar B (2007) Introduction to statistical relational learning. MIT, Cambridge

Golub TR, Slonim DK, Tamayo P, Huard C, Gaasenbeek M, Mesirov JP et al (1999) Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. Science 286:531–537

Hardin J, Waddell M, Page CD, Zhan F, Barlogie B, Shaugh-nessy J et al (2004) Evaluation of multiple models to distinguish closely related forms of disease using DNA microarray data: an application to multiple myeloma. Stat Appl Gene Mol Biol 3(1):1018

Jain AN, Dietterich TG, Lathrop RH, Chapman D, Critchlow RE, Bauer BE et al (1994) Compass: a shape-based machine learning tool for drug design. Aided Mol Des 8(6):635–652

Jones KE, Reiser FM, Bryant PGK, Muggleton CH, Kell S, King DB et al (2004) Functional genomic hypothesis generation and experimentation by a robot scientist. Nature 427:247–252

KDD Cup (2001) http://pages.cs.wisc.edu/-dpage/kddcup2001/

Klösgen W (2002) Handbook of data mining and knowledge discovery, chapter 16.3: subgroup discovery. Oxford University Press, New York

Listgarten J, Damaraju S, Poulin B, Cook L, Dufour J, Driga A et al (2004) Predictive models for breast cancer susceptibility from multiple single nucleotide polymorphisms. Clin Cancer Res 10:2725–2737

Mardis ER (2006) Anticipating the 1,000 dollar genome. Genome Biol 7(7):112

Martin YC, Bures MG, Danaher EA, DeLazzer J, Lico II, Pavlik PA (1993) A fast new approach to pharmacophore mapping and its application to dopaminergic and benzodiazepine agonists. J Comput Aided Mol Des 8:751–758

McCarty C, Wilke RA, Giampietro PF, Wesbrook SD, Caldwell MD (2005) Personalized medicine research project (PMRP): design, methods and recruitment for a large population-based biobank. Personal Med 2:49–79

Molla M, Waddell M, Page D, Shavlik J (2004) Using machine learning to design and interpret gene expression microarrays. AI Mag 25(1): 23–44

Muggleton S, De Raedt L (1994) Inductive logic programming: theory and methods. J Log Program 19(20):629–679

Noto K, Craven M (2006) A specialized learner for inferring structured cis-regulatorymodules. BMC Bioinform 7(528). doi:10.1186/1471-2105-7-528

Oliver SG, Young M, Aubrey W, Byrne E, Liakata M, Markham M et al (2009) The automation of science. Science 324:85–89

Ong I, Glassner J, Page D (2002) Modelling regulatory pathways in E.coli from time series expression profiles. Bioinformatics 18:241S–248S

Pe'er D, Regev A, Elidan G, Friedman N (2001) Inferring subnetworks from perturbed expression profiles. Bioinformatics 17:215–224

Perou C, Jeffrey S, Van De Rijn M, Rees CA, Eisen MB, Ross, DT et al (1999) Distinctive gene expression patterns in human mammary epithelial cells and breast cancers. Proc Natl Acad Sci 96:9212–9217

Petricoin EF III, Ardekani AM, Hitt BA, Levine PJ, Fusaro VA, Steinberg SM et al (2002) Use of proteomic patterns in serum to identify ovarian cancer. Lancet 359:572–577

Rost B, Sander C (1993) Prediction of protein secondary structure at better than 70 accuracy. J Mol Biol 232:584–599

Segal E, Pe'er D, Regev A, Koller D, Friedman N (2005) Learning module networks. J Mach Learn Res 6:557–588

Spatola A, Page D, Vogel D, Blondell S, Crozet Y (1999) Can machine learning and combinatorial chemistry co-exist? In: Proceedings of the American peptide symposium, Minneapolis. Kluwer Academic

Srinivasan A (2001) The aleph manual. http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/

Storey JD, Tibshirani R (2003) Statistical significance for genome-wide studies. Proc Natl Acad Sci 100:9440–9445

The International Warfarin Pharmacogenetics Consortium (2009) Estimation of the Warfarin dose with clinical and pharmacogenetic data. N Engl J Med 360:753–764

Tucker A, Vinciotti V, Hoen PAC, Liu X, Famili AF (2005) Bayesian network classifiers for time-series microarray data. Adv Intell Data Anal VI 3646:475–485

Van't Veer LL, Dai H, van de Vijver MM, He Y, Hart A, Mao M et al (2002) Gene expression profiling predicts clinical outcome of breast cancer. Nature 415:530–536

Waddell M, Page D, Shaughnessy J Jr (2005) Predicting cancer susceptibility from single-nucleotide polymorphism data: a case study in multiple myeloma. In: BIOKDD'05: proceedings of the fifth international workshop on bioinformatics, Chicago

Wrobel S (1997) An algorithm for multi-relational discovery of subgroups. In: European symposium on principles of KDD, Trondheim. Lecture notes in computer science. Springer, pp 78–87

Zhang X, Mesirov JP, Waltz DL (1992) Hybrid system for protein secondary structure prediction. J Mol Biol 225:81–92

Zou M, Conzen SD (2005) A new dynamic Bayesian network approach for identifying gene regulatory networks from time course microarray data. Bioinformatics 21:71–79

Blog Mining

Blog mining is the application of data mining (in particular, Web mining) techniques on blogs, adapted to the content, format, and language of the medium blog. A *blog* is a (more or less) frequently updated publication on the Web, sorted in (usually reverse) chronological order of the constituent blog posts. As in other areas of the Web, mining is applied to the content of blogs, to the various types of links between blogs, and to blog-related behavior. The latter comprises blog authoring including link setting, blog reading and commenting, and querying (often in blog search engines). For more details on blogs and on mining them, see ▶ text mining for news and blogs analysis.

Boltzmann Machines

Geoffrey Hinton
University of Toronto, Toronto, ON, Canada

Definition

A Boltzmann machine is a network of symmetrically connected, neuron-like units that make stochastic decisions about whether to be on or off. Boltzmann machines have a simple learning algorithm (Hinton and Sejnowski 1983) that allows them to discover interesting features that represent complex regularities in the training data. The learning algorithm is very slow in networks with many layers of feature detectors, but it is fast in "restricted Boltzmann machines" that have a single layer of feature detectors. Many hidden layers can be learned efficiently by composing restricted Boltzmann machines, using the feature activations of one as the training data for the next.

Boltzmann machines are used to solve two quite different computational problems. For a search problem, the weights on the connections are fixed and are used to represent a cost function. The stochastic dynamics of a Boltzmann machine then allow it to sample binary state vectors that have low values of the cost function. For a learning problem, the Boltzmann machine is shown a set of binary data vectors, and it must learn to generate these vectors with high probability. To do this, it must find weights on the connections so that relative to other possible binary vectors, the data vectors have low values of the cost function. To solve a learning problem, Boltzmann machines make many small updates to their weights, and each update requires them to solve many different search problems.

Motivation and Background

The brain is very good at settling on a sensible interpretation of its sensory input within a few hundred milliseconds, and it is also very good, over a much longer timescale, at learning the code that is used to express its interpretations. It achieves both the settling and the learning using spiking neurons which, over a period of a few milliseconds, have a state of 1 or 0. These neurons have intrinsic noise caused by the quantal release of vesicles of neurotransmitter at the synapses between the neurons.

Boltzmann machines were designed to model both the settling and the learning and were based on two seminal ideas that appeared in 1982. Hopfield (1982) showed that a neural network composed of binary units would settle to a minimum of a simple, quadratic energy function provided that the units were updated asynchronously and the pairwise connections between units were symmetrically weighted. Kirkpatrick et al. (1983) showed that systems that were settling to energy minima could find deeper minima if noise was added to the update rule so that the system could occasionally increase its energy to escape from poor local minima.

Adding noise to a Hopfield net allows it to find deeper minima that represent more probable interpretations of the sensory data. More significantly, by using the right kind of noise, it is possible to make the log probability of finding the system in a particular global configuration be a linear function of its energy. This makes it possible to manipulate log probabilities by manipulating energies, and since energies are simple local functions of the connection weights, this leads to a simple, local learning rule.

Structure of Learning System

The learning procedure for updating the connection weights of a Boltzmann machine is very simple, but to understand why it works, it is first necessary to understand how a Boltzmann machine models a probability distribution over a set of binary vectors and how it samples from this distribution.

The Stochastic Dynamics of a Boltzmann Machine

When unit i is given the opportunity to update its binary state, it first computes its total input, x_i, which is the sum of its own bias, b_i, and the weights on connections coming from other active units:

$$x_i = b_i + \sum_j s_j w_{ij} \tag{1}$$

where w_{ij} is the weight on the connection between i and j and s_j is 1 if unit j is on and 0, otherwise. Unit i then turns on with a probability given by the logistic function

$$\text{prob}(s_i = 1) = \frac{1}{1 + \mathrm{e}^{-x_i}} \tag{2}$$

If the units are updated sequentially in any order that does not depend on their total inputs, the network will eventually reach a Boltzmann distribution (also called its equilibrium or stationary distribution) in which the probability of a state vector, **v**, is determined solely by the "energy" of that state vector relative to the energies of all possible binary state vectors:

$$P(\mathbf{v}) = \mathrm{e}^{-E(\mathbf{v})} / \sum_{\mathbf{u}} \mathrm{e}^{-E(\mathbf{u})} \tag{3}$$

As in Hopfield nets, the energy of state vector **v** is defined as

$$E(\mathbf{v}) = -\sum_i s_i^{\mathbf{v}} b_i - \sum_{i<j} s_i^{\mathbf{v}} s_j^{\mathbf{v}} w_{ij} \tag{4}$$

where $s_i^{\mathbf{v}}$ is the binary state assigned to unit i by state vector **v**.

If the weights on the connections are chosen so that the energies of state vectors represent the cost of those state vectors, then the stochastic dynamics of a Boltzmann machine can be viewed as a way of escaping from poor local optima while searching for low-cost solutions. The total input to unit i, x_i, represents the difference in energy depending on whether the unit is off or on, and the fact that unit i occasionally turns on even if x_i is negative means that the energy can occasionally increase during the search, thus allowing the search to jump over energy barriers.

The search can be improved by using simulated annealing. This scales down all of the weights and energies by a factor, T, which is analogous to the temperature of a physical system. By reducing T from a large initial value to a small final value, it is possible to benefit from the fast equilibration at high temperatures and still have a final equilibrium distribution that makes low-cost solutions much more probable than high-cost ones. At a temperature of 0, the update rule becomes deterministic and a Boltzmann machine turns into a Hopfield network.

Learning in Boltzmann Machines Without Hidden Units

Given a training set of state vectors (the data), the learning consists of finding weights and biases (the parameters) that make those state vectors good. More specifically, the aim is to find weights and biases that define a Boltzmann distribution in which the training vectors have high probability. By differentiating (3) and using the fact that

$$\partial E(\mathbf{v}) / \partial w_{ij} = -s_i^{\mathbf{v}} s_j^{\mathbf{v}} \tag{5}$$

it can be shown that

$$\left\langle \frac{\partial \log P(\mathbf{v})}{\partial w_{ij}} \right\rangle_{\text{data}} = \langle s_i s_j \rangle_{\text{data}} - \langle s_i s_j \rangle_{\text{model}} \tag{6}$$

where $\langle\cdot\rangle_{\text{data}}$ is an expected value in the data distribution and $\langle\cdot\rangle_{\text{model}}$ is an expected value when the Boltzmann machine samples state vectors from its equilibrium distribution at a temperature of 1. To perform gradient ascent in the log probability that the Boltzmann machine would generate the observed data when sampling from its equilibrium distribution, w_{ij} is incremented by a small learning rate times the RHS of (6). The learning rule for the bias, b_i, is the same as (6), but with s_j omitted.

If the observed data specifies a binary state for every unit in the Boltzmann machine, the learning problem is convex: there are no nonglobal optima in the parameter space. However, sampling from $\langle\cdot\rangle_{\text{model}}$ may involve overcoming energy barriers in the binary state space.

Learning with Hidden Units

Learning becomes much more interesting if the Boltzmann machine consists of some "visible" units whose states can be observed and some "hidden" units whose states are not specified by the observed data. The hidden units act as latent variables (features) that allow the Boltzmann machine to model distributions over visible state vectors that cannot be modeled by direct pairwise interactions between the visible units. A surprising property of Boltzmann machines is that, even with hidden units, the learning rule remains unchanged. This makes it possible to learn binary features that capture higher-order structure in the data. With hidden units, the expectation $\langle s_i s_j \rangle_{\text{data}}$ is the average, over all data vectors, of the expected value of $s_i s_j$ when a data vector is clamped on the visible units, and the hidden units are repeatedly updated until they reach equilibrium with the clamped data vector.

It is surprising that the learning rule is so simple because $\partial \log P(\mathbf{v})/\partial w_{ij}$ depends on all the other weights in the network. Fortunately, the locally available difference in the two correlations in (6) tells w_{ij} everything it needs to know about the other weights. This makes it unnecessary to explicitly propagate error derivatives, as in the backpropagation algorithm.

Different Types of Boltzmann Machine

The stochastic dynamics and the learning rule can accommodate more complicated energy functions (Sejnowski 1986). For example, the quadratic energy function in (4) can be replaced by an energy function that has typical term $s_i s_j s_k w_{ijk}$. The total input to unit i that is used in the update rule must then be replaced by

$$x_i = b_i + \sum_{j<k} s_j s_k w_{ijk}. \tag{7}$$

The only change in the learning rule is that $s_i s_j$ is replaced by $s_i s_j s_k$.

Boltzmann machines model the distribution of the data vectors, but there is a simple extension, the "conditional Boltzmann machine" for modeling conditional distributions (Ackley et al. 1985). The only difference between the visible and the hidden units is that when sampling $\langle s_i s_j \rangle_{\text{data}}$, the visible units are clamped and the hidden units are not. If a subset of the visible units are also clamped when sampling $\langle s_i s_j \rangle_{\text{model}}$, this subset acts as "input" units and the remaining visible units act as "output" units. The same learning rule applies, but now it maximizes the log probabilities of the observed output vectors conditional on the input vectors.

Instead of using units that have stochastic binary states, it is possible to use "mean field" units that have deterministic, real-valued states between 0 and 1, as in an analog Hopfield net. Equation (2) is used to compute an "ideal" value for a unit's state, given the current states of the other units, and the actual value is moved toward the ideal value by some fraction of the difference. If this fraction is small, all the units can be updated in parallel. The same learning rules can be used by simply replacing the stochastic, binary values by the deterministic real values (Peterson and Anderson 1987), but the learning algorithm is hard to justify and the mean field nets have problems in modeling multimodal distributions.

The binary stochastic units used in Boltzmann machines can be generalized to "softmax" units that have more than two discrete values, Gaussian units whose output is simply their total input plus Gaussian noise, binomial units, Poisson units, and any other type of unit that falls in the exponential family (Welling et al. 2005). This family is characterized by the fact that the adjustable parameters have linear effects on the log probabilities. The general form of the gradient required for learning is simply the change in the sufficient statistics caused by clamping data on the visible units.

The Speed of Learning

Learning is typically very slow in Boltzmann machines with many hidden layers because large networks can take a long time to approach their equilibrium distribution, especially when the weights are large and the equilibrium distribution is highly multimodal, as it usually is when the visible units are unclamped. Even if samples from the equilibrium distribution can be obtained, the learning signal is very noisy because it is the difference of two sampled expectations. These difficulties can be overcome by restricting the connectivity, simplifying the learning algorithm, and learning one hidden layer at a time.

Restricted Boltzmann Machines

A restricted Boltzmann machine (Smolensky 1986) consists of a layer of visible units and a layer of hidden units with no visible-visible or hidden-hidden connections. With these restrictions, the hidden units are conditionally independent given a visible vector, so unbiased samples from $\langle s_i s_j \rangle_{\text{data}}$ can be obtained in one parallel step. To sample from $\langle s_i s_j \rangle_{\text{model}}$ still requires multiple iterations that alternate between updating all the hidden units in parallel and updating all of the visible units in parallel. However, learning still works well if $\langle s_i s_j \rangle_{\text{model}}$ is replaced by $\langle s_i s_j \rangle_{\text{reconstruction}}$ which is obtained as follows:

1. Starting with a data vector on the visible units, update all of the hidden units in parallel.
2. Update all of the visible units in parallel to get a "reconstruction."
3. Update all of the hidden units again.

This efficient learning procedure approximates gradient descent in a quantity called "contrastive divergence" and works well in practice (Hinton 2002).

Learning Deep Networks by Composing Restricted Boltzmann Machines

After learning one hidden layer, the activity vectors of the hidden units, when they are being driven by the real data, can be treated as "data" for training another restricted Boltzmann machine. This can be repeated to learn as many hidden layers as desired. After learning multiple hidden layers in this way, the whole network can be viewed as a single, multilayer generative model, and each additional hidden layer improves a lower bound on the probability that the multilayer model would generate the training data (Hinton et al. 2006).

Learning one hidden layer at a time is a very effective way to learn deep neural networks with many hidden layers and millions of weights. Even though the learning is unsupervised, the highest-level features are typically much more useful for classification than the raw data vectors. These deep networks can be fine-tuned to be better at classification or dimensionality reduction using the backpropagation algorithm (Hinton and Salakhutdinov 2006). Alternatively, they can be fine-tuned to be better generative models using a version of the "wake-sleep" algorithm Hinton et al. (2006).

Relationships to Other Models

Boltzmann machines are a type of Markov random field (see ▶ Graphical Models), but most Markov random fields have simple, local interaction weights which are designed by hand rather than being learned. Boltzmann machines are also like Ising models, but Ising models typically use random or hand-designed interaction weights. The search procedure for Boltzmann machines is an early example of Gibbs sampling, a ▶ Markov chain Monte Carlo method which was invented independently (Geman and Geman 1984) and was also inspired by simulated annealing.

Boltzmann machines are a simple type of undirected graphical model. The learning algorithm for Boltzmann machines was the first learning algorithm for undirected graphical models with hidden variables (Jordan 1998). When restricted Boltzmann machines are composed to learn a deep network, the top two layers of the resulting graphical model form an undirected Boltzmann machine, but the lower layers form a directed acyclic graph with directed connections from higher layers to lower layers (Hinton et al. (2006)).

Conditional random fields (Lafferty et al. 2001) can be viewed as simplified versions of higher-order, conditional Boltzmann machines in which the hidden units have been eliminated. This makes the learning problem convex but removes the ability to learn new features.

Recommended Reading

Ackley D, Hinton G, Sejnowski T (1985) A Learning algorithm for Boltzmann machines. Cognit Sci 9(1):147–169

Geman S, Geman D (1984) Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. IEEE Trans Pattern Anal Mach Intell 6(6):721–741

Hopfield JJ (1982) Neural networks and physical systems with emergent collective computational abilities. Proc Natl Acad Sci U S A 79:2554–2558

Hinton GE (2002) Training products of experts by minimizing contrastive divergence. Neural Comput 14(8):1711–1800

Hinton GE, Osindero S, Teh YW (2006) A fast learning algorithm for deep belief nets. Neural Comput 18:1527–1554

Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313:504–507

Hinton GE, Sejnowski TJ (1983) Optimal perceptual inference. In: Proceedings of the IEEE conference on computer vision and pattern recognition, Washington, DC, pp 448–453

Jordan MI (1998) Learning in graphical models. MIT, Cambridge

Kirkpatrick S, Gelatt DD, Vecci MP (1983) Optimization by simulated annealing. Science 220(4598):671–680

Lafferty J, McCallum A, Pereira F (2001) Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proceedings of the 18th international conference on machine learning, Williamstown, pp 282–289. Morgan Kaufmann, San Francisco

Peterson C, Anderson JR (1987) A mean field theory learning algorithm for neural networks. Complex Syst 1(5):995–1019

Sejnowski TJ (1986) Higher-order Boltzmann machines. AIP Conf Proc 151(1):398–403

Smolensky P (1986) Information processing in dynamical systems: foundations of harmony theory. In: Rumelhart DE, McClelland JL (eds) Parallel distributed processing. Foundations, vol 1. MIT, Cambridge, pp 194–281 Press.

Welling M, Rosen-Zvi M, Hinton GE (2005) Exponential family harmoniums with an application to information retrieval. In: Lawrence K. Saul, Yair Weiss and Leon Bottou (eds) Advances in neural information processing systems, vol 17. MIT, Cambridge, pp 1481–1488

Boosting

Boosting is a family of ▸ ensemble learning methods. The Boosting framework is an answer to a question posed on whether two complexity classes of learning problems are equivalent: *strongly learnable*, and *weakly learnable*. The Boosting framework is a proof by construction that the answer is positive, they are equivalent. The framework allows a "weak" model, only slightly better than random guessing, to be *boosted* into an arbitrarily accurate *strong* model. ▸ Adaboostis the most well known and successful of the Boosting family, though there exist many variants specialized for particular tasks, such as cost-sensitive and noise-tolerant versions. See ▸ ensemble learning for full details.

Bootstrap Sampling

Definition

Bootstrap sampling is a process for creating a distribution of datasets out of a single dataset. It is used in the ▸ ensemble learning algorithm ▸ Bagging. It can also be used in ▸ algorithm evaluation to create a distribution of training sets from which to estimate properties of an algorithm.

Recommended Reading

Davison AC, Hinkley D (2006) Bootstrap methods and their applications, 8th edn. Cambridge series in statistical and probabilistic mathematics. Cambridge University Press, Cambridge

Bottom Clause

Synonyms

Saturation; Starting clause

Definition

The bottom clause is a notion from the field of ▶ inductive logic programming. It is used to refer to the most specific hypothesis covering a particular example when learning from entailment. When ▶ learning from entailment, a hypothesis H covers an example e relative to the background theory B if and only if $B \wedge H \models e$, that is, B together with H ▶ entails the example e. The bottom clause is now the most specific clause satisfying this relationship w.r.t the background theory B and a given example e.

For instance, given the background theory B

```
bird :- blackbird.
bird :- ostrich.
```

and the example e:

```
flies :- blackbird, normal.
```

the bottom clause is H

```
flies :- bird, blackbird, normal.
```

The bottom clause can be used to constrain the search for clauses covering the given example because all clauses covering the example relative to the background theory should be more general than the bottom clause. The bottom clause can be computed using inverse entailment.

Cross-References

▶ Entailment
▶ Inductive Logic Programming
▶ Inverse Entailment
▶ Logic of Generality

Bounded Differences Inequality

▶ McDiarmid's Inequality

BP

▶ Backpropagation

Breakeven Point

More accurately described as *precision–recall* BEP, it is an evaluation measure originally introduced in the field of information retrieval to evaluate retrieval systems that return a list of documents ordered by their supposed relevance to the user's information need (see also ▶ Document Classification). It can also be used to evaluate any classification model f that addresses a two-class classification problem but outputs real-valued predictions $f(x)$ instead of binary ones. To use such a classifier in practice, one would select a threshold θ and predict an instance x to be positive if $f(x) > \theta$ and negative otherwise. Thus, the ▶ precision and ▶ recall of this system depend on the choice of the threshold θ. A lower threshold means higher recall, but usually also lower precision. At some point (when the number of instances predicted to be positive is the same as the actual number of positive instances), precision and recall are equal; this value of precision and recall is known as the *precision–recall BEP*. It is a useful measure of the quality of our classifier because it gives us guidance into what sort of tradeoffs are available to the user of such a classifier via the choice of threshold: if we want a precision above the BEP, we must accept that our recall will be below the BEP, and vice versa. A different meaning of the term "breakeven point" is sometimes used in ROC (▶ ROC Analysis), where the *ROC breakeven* is defined as the point where the true positive rate and the false positive rate sum to 1; smaller values of the ROC breakeven are better than larger ones. Informally, the ROC breakeven measures how close the ROC curve gets to the "ROC sweet spot" in the top left corner (where the ▶ true positive rate is 1 and the ▶ false positive rate is 0).

C

Candidate-Elimination Algorithm

Mitchell's, (1982; 1997) candidate-elimination algorithm performs a bidirectional search in the ▸ hypothesis space. It maintains a set, S, of most specific hypotheses that are consistent with the training data and a set, G, of most general hypotheses consistent with the training data. These two sets form two boundaries on the version space.

Recommended Reading

Mitchell TM (1982) Generalization as search Artif Intell 18(2):203–226

Mitchell TM (1997) Machine learning. McGraw-Hill, New York

Cannot-Link Constraint

A pairwise constraint between two items indicating that they should be placed into different clusters in the final partition.

Cascade Correlation

Thomas R. Shultz[1] and Scott E. Fahlman[2]
[1]McGill University, Montréal, QC, Canada
[2]Carnegie Mellon University, Pittsburgh,
PA, USA

Synonyms

Cascor; CC

Definition

Cascade–correlation (often abbreviated as "Cascor" or "CC") is a supervised learning algorithm for artificial neural networks. It is related to the back-propagation algorithm ("backprop"). CC differs from backprop in that a CC network begins with no hidden units and then adds units one by one, as needed during learning.

Each new hidden unit is trained to correlate with residual error in the network built so far. When it is added to the network, the new unit is frozen, in the sense that its input weights are fixed. The hidden units form a *cascade*: each

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

new unit receives weighted input from all the original network inputs and from the output of every previously created hidden unit. This cascading creates a network that is as deep as the number of hidden units. Stated another way, the CC algorithm is capable of efficiently creating complex, higher-order nonlinear basis functions – the hidden units – which are then combined to form the desired outputs.

The result is an algorithm that learns complex input/output mappings very fast compared to backprop and that builds a multilayer network structure that is customized for the problem at hand.

Motivation and Background

Cascade–correlation was designed (Fahlman and Lebiere 1990) to address two well-known problems with the popular back-propagation algorithm ("backprop"). First, a backprop user has to guess what network structure – the number of hidden layers and the number of units in each layer – would be best for a given learning problem. If the network is too small or too shallow, it won't solve the problem; if it is too large or too deep, training is very slow, and the network is prone to overfitting the training data. Because there is no reliable way to choose a good structure before training begins, most backprop users have to train many different structures before finding one that is well matched to the task.

Second, even if a backprop user manages to choose a good network structure, training is generally very slow. That is particularly true in networks with many hidden units or with more than one hidden layer. One cause of slow learning in backprop is the use of a uniform learning rate parameter for updating network weights. This problem was addressed with the Quickprop algorithm (Fahlman 1988), an approximation to Newton's method that adapts the learning rate for each weight parameter depending on the first two derivatives of the local error surface. Quickprop improved learning speed, sometimes dramatically, but learning was still too slow in large or deep networks.

Another cause of slow learning in backprop is the "herd effect" (Fahlman and Lebiere 1990). If the solution to a network problem requires, say, 30 hidden units, each of these units must be trained to do a different job – that is, to compute a different nonlinear basis function. Each hidden unit starts with a different and randomly chosen set of input weights; but if the units are all trained at once, they all see the same error signal. There is no central authority telling each unit to do a separate job, so they tend to drift toward the same part of parameter space, forming a herd that moves around together. Eventually, the units may drift apart and begin to differentiate, but there is nothing to compel this, so the process is slow and unreliable. Usually, in selecting an initial topology for a backprop net, it is necessary to include many extra hidden units to increase the odds that each job will be done by some unit.

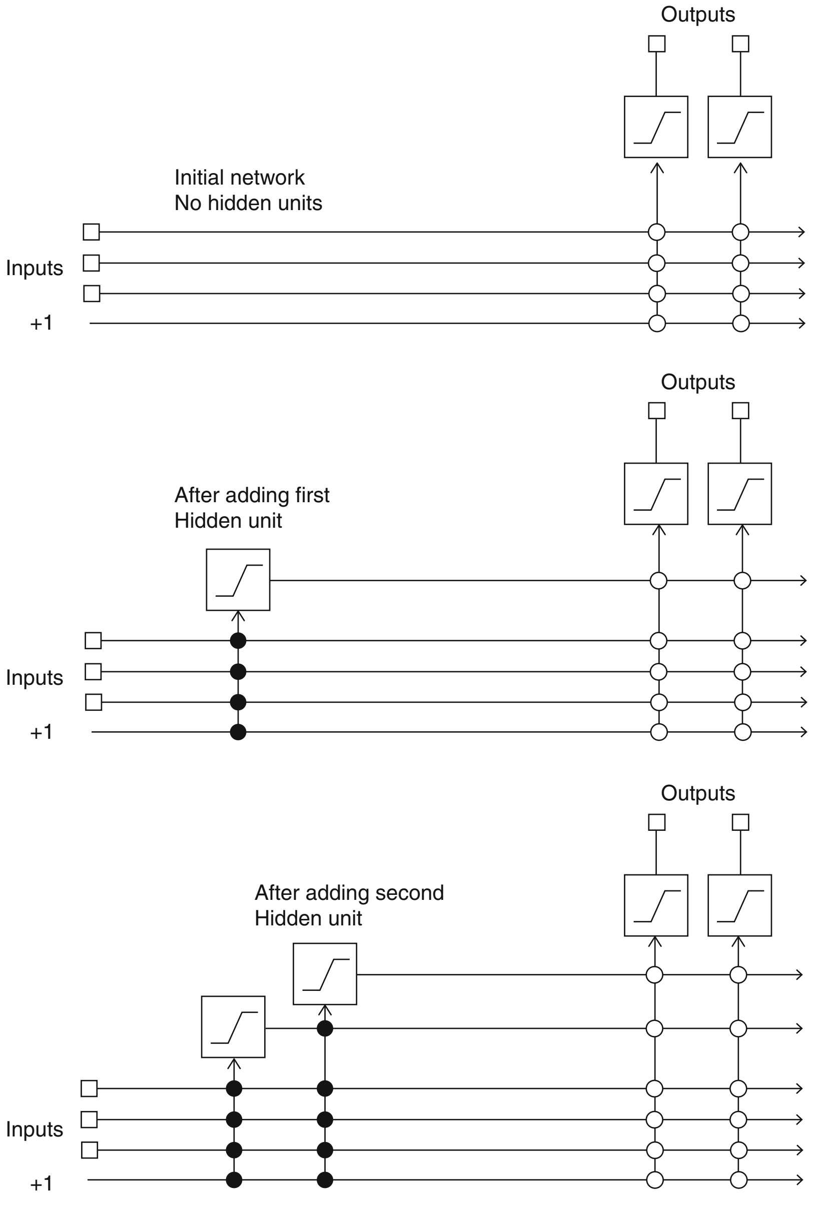
CC addresses this problem by introducing and training hidden units one by one. Each hidden unit sees a strong, clear error gradient, not confused by the simultaneous movement of other hidden units. A new hidden unit can thus move quickly and decisively to a position in parameter space where it can perform a useful function, reducing the residual error. One by one, Cascor-hidden units take up distinct jobs, instead of milling about together competing to do the same job.

Structure of Learning System

The Algorithm

The CC architecture is illustrated in Fig. 1. It begins with some inputs and one or more output units, but no hidden units. The numbers of inputs and outputs are dictated by the problem. As in backprop, the output units generally have a sigmoid activation function, but could alternatively have a linear activation function. Every input is connected to every output unit by a connection with an adjustable weight. There is also a *bias* input, permanently set to +1.

Hidden units are added to the network one by one. Each new hidden unit receives a weighted connection from each of the network's original



Cascade Correlation, Fig. 1 The Cascade–correlation (CC) architecture, as new hidden units are added. *Black circles* are frozen connection weights; *white circles* are weights trained during output training phase. The *vertical lines* sum all incoming activation

inputs and also from every existing hidden unit. Each new unit therefore adds a new single-unit layer to the network. This makes it possible to create high-order nonlinear feature detectors, customized for the problem at hand.

As noted, learning begins without hidden units. The direct input–output connections are trained as well as possible over the entire set of training examples, using Quickprop. At some point, this training approaches an asymptote. When no significant error reduction has occurred after a certain number of training cycles, this output phase is terminated and there is a shift to input phase to recruit a new hidden unit, using the unit-creation algorithm to be described. The new unit is added to the net, its input weights are frozen, and all the output weights are once again trained using Quickprop. This cycle repeats until the error is acceptably small, in the sense that all network outputs for all training patterns are within a specified threshold of their target values.

To create a new hidden unit, input phase begins with several *candidate units* that receive trainable input connections from all of the network inputs and from all existing hidden units. The outputs of these candidates are not yet connected to the network. There are a number of passes over the examples of the training set, adjusting the candidate unit's input weights after each pass. The goal of these adjustments, using Quickprop, is to maximize the correlation between each candidate's output and the residual error.

When these correlation measures show no further significant improvement, input phase stops, the best-correlating candidate's input weights are frozen, and that unit is installed in the network. The remaining candidates are discarded and the algorithm then retrains the output weights, making use of this new feature as well as all the old ones. As the new unit's output correlates well with some component of the residual error, its output weights can be quickly adjusted to reduce that component. So after adding each new hidden unit, the network's residual error should be smaller than before.

Using several candidates, each with differently initialized input weights, greatly reduces the chances of installing a bad hidden unit that gets the network stuck in a local optimum far from the global optimum value. All candidates receive the same input signals and see the same residual error for each training pattern. Because they do not interact with one another or affect the network during training, these candidates can be trained in parallel. In a pool of four to eight candidates, there are almost always several high-quality candidates with nearly equal correlation values.

Hidden units continue to be recruited until network error reaches an acceptable level, or until cross-validation signals a stop. Because only a single layer of weights is adjusted at a time, rather than back-propagating an error signal through several layers of shifting units, CC training proceeds very quickly.

Performance

CC is designed to produce a network just large enough to solve the problem and to do so much faster than backprop and related algorithms. In many reported cases that require hidden units, CC learns the desired behavior 10–100 times faster than standard backprop (Fahlman and Lebiere 1990). One striking example of this is the *two-spirals problem*, an artificial benchmark designed to be very difficult for neural networks with sigmoid units. At the time CC was developed, the best known backprop solutions for two spirals required a network with three hidden layers of five units each. CC typically solves this problem with 12 hidden units and has found solutions with as few as nine hidden units. In terms of runtime, CC training was about 50 times faster than standard backprop and 23 times faster than Quickprop used within a static network.

Variants of Cascade Correlation

Flat Cascade Correlation

In standard CC, each new hidden unit receives inputs from every existing unit, so the net becomes one level deeper every time a unit is added. This is a powerful mechanism, creating increasingly complex feature detectors as the network learns. But sometimes this added depth is not required for the problem, creating a very deep network that performs no better than a shallow one. The resulting network might have more weights than are required for the problem, raising concern about overfitting. Another concern was that the cascaded nonlinearity of CC might also compromise generalization. To address these concerns, a flat variant of Cascor adds new recruited units onto a single layer (i.e., cascaded connections are eliminated), limiting the depth of the network and eliminating all cascaded weights between hidden units.

Comparison of flat to standard CC on generalization in particular learning problems yielded inconsistent results, but a more problem–neutral, student–teacher approach found no generalization differences between flat and standard versions of CC (Dandurand et al. 2007). Here, flat and standard student CC networks learned the input–output mappings of other randomly initialized flat and standard CC teacher networks, where task complexity was systematically manipulated. Both standard and flat CC student networks learned and generalized well on problems of varying complexity. In low-complexity tasks, there were no significant performance differences between flat and standard CC student networks. For high-complexity tasks, flat CC student networks required fewer connection weights and learned with less computational cost than did standard CC student networks.

Sibling–Descendant Cascade–Correlation (SDCC)

SDCC (Baluja and Fahlman 1994) provides a more general solution to the problem of network depth. In the candidate pool, there are two kinds of candidate units: *descendant* units that receive inputs from all existing hidden units and *sibling* units that receive the same inputs as the deepest hidden units in the current net. When the time comes to choose a winning candidate, the candidate with the best correlation wins, but there is a slight preference for sibling units. So unless a descendant unit is clearly superior, a sibling unit will be recruited, making the active network larger, but not deeper. In problems where standard CC produces a network with 15 or 20 hidden units and an equal number of layers, SDCC often produces a network with only two or three hidden layers.

Recurrent Cascade–Correlation (RCC)

Standard CC produces a network that maps its *current* inputs to outputs. The network has no memory of recent inputs, so this architecture is not able to learn to recognize a sequence of inputs. In the RCC algorithm, each candidate and hidden unit takes the same inputs as in standard CC, but it also takes an additional input: the unit's own previous output, delayed by one time interval (Fahlman 1991). The weight on this time-delayed input is trained by the same algorithm as all the other inputs.

This delayed loop gives RCC networks a way of remembering past inputs and internal states, so they can learn to recognize sequences of input patterns. In effect, the architecture builds a finite-state machine tailored specifically to recognize the pattern sequences in the training set. For example, an RCC net learned to recognize characters in Morse code.

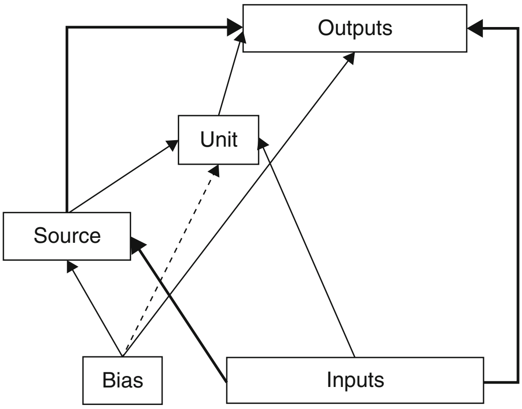
Knowledge-Based Cascade–Correlation (KBCC)

KBCC is a variant that can recruit previously-learned networks or indeed any differentiable function, in competition with single hidden units (Shultz and Rivest 2001; Shultz et al. 2007). The recruit is the candidate whose output correlates best with residual network error, just as in ordinary CC. The candidate pool usually has a number of randomly initialized sigmoid units and a number of candidate source networks, i.e., networks previously trained on other tasks. The input weights to multiple copies of the source networks are usually randomly initialized to improve optimization. Of these copies, one is typically connected with an identity matrix with off-diagonal zeros, to enable quick learning of the target task when exact knowledge is available. A hypothetical KBCC network is shown in Fig. 2.

Software

Most CC algorithms are available in a variety of formats and languages, including:

CASCOR: Lisp and C implementations of cascade–correlation
http://www.cs.cmu.edu/afs/cs/project/ai-repository/ ai/areas/neural/systems/cascor/0.html
Free Lisp and C implementations of cascade–correlation
Cascade Neural Network Simulator
http://www.cs.cmu.edu/~sef/sefSoft.htm
A public domain C program that implements cascade–correlation and recurrent cascade–correlation, plus experimental versions of cascade 2 and recurrent cascade 2
LNSC cascade–correlation simulator applet



Cascade Correlation, Fig. 2 Hypothetical knowledge-based cascade–correlation (KBCC) network that has recruited a source network and then a sigmoid unit, each installed on a separate layer. The *dashed line* represents a single connection weight, *thin solid lines* represent weight vectors, and *thick solid lines* represent weight matrices

http://www.psych.mcgill.ca/perpg/fac/shultz/cdp/lnsc_applet.htm
A Java applet allowing direct comparisons of cascade–correlation and back-propagation algorithms on some benchmark problems, also permitting entry of text-edited custom training and test patterns.
LNSC Java Code Library
http://www.lnsclab.org/
Free compiled Java versions of BP, CC, SDCC, and KBCC neural network software, along with a tutorial

Applications

CC

Partly because of its ability to grow its own networks and build new learning on top of existing knowledge, CC has been used to simulate many phenomena in cognitive development. These characteristics embody the constructivism that developmental psychologists often discussed but did not formulate precisely. Simulations are typically evaluated by how well they capture the various psychological phenomena that characterize a particular domain.

The balance-scale task involves presenting a child with a rigid beam balanced on a fulcrum with pegs spaced at equal intervals to the left and right of the fulcrum. A number of identical weights are placed on a peg on the left side and a peg on the right side, and the child is asked to predict which side will descend when the beam is released from its moorings. CC networks passed through the stages observed with children and captured the so-called torque difference effect, the tendency to do better on problems with large absolute torque differences than on problems with small torque differences (Shultz et al. 1994; Shultz and Takane 2007).

The conservation task presents a child with two quantities of objects that the child judges to be equal and then transforms one set in a way that either changes that relationship or conserves it. CC networks captured four important conservation regularities (Shultz 1998):

1. A shift from nonconservation to conservation beliefs
2. A sudden spurt in performance during acquisition
3. Emergence of correct conservation judgments for small quantities before larger quantities
4. Young children's choice of the longer row as having more items than the shorter row

Analysis of network solutions at various points in development revealed a gradual shift from perceptual (how the sets of items look) to cognitive (whether or not the transformation changed a quantity) solutions, similar to what had been found with children.

The seriation task requires a child to order a disordered collection of sticks of different lengths. CC networks passed through the four stages seen in children (total failure, partial sort, trial-and-error sort, and systematic sort) and captured the tendency for sets with smaller differences to be more difficult to sort (Mareschal and Shultz 1999). Analysis of network solutions revealed early success at the short end of the series that was gradually extended to the longer end, as in children.

The transitivity problem typically also employs sticks of different length. Here the child is trained on all pairs of sticks that are adjacent in length and then is asked to infer the relative length of untrained pairs. Five psychological regularities were captured when CC networks were trained to compare the relative sizes of adjacent pairs (Shultz and Vogel 2004):

1. Learning short or long adjacent pairs before adjacent pairs of medium length.
2. Faster inferences with pairs farther apart in length than with pairs close together in length, an effect that diminished with age. A constraint-satisfaction network module simulated reaction times by inputting the output of a CC network and settling over time cycles into a low-energy solution that satisfied the constraints supplied by connection weights and inputs, effectively cleaning up the output of the CC network.
3. Faster inferences with pairs containing the shortest or longest stick.
4. Faster inferences when the expression used in the question (e.g., shorter) is compatible with an end stick (e.g., the shortest stick) in the compared pair than when the question term (e.g., shorter) is incompatible with an end stick (e.g., the longest stick) in the compared pair.
5. Older children learned adjacent pairs faster and made inference comparisons faster and more accurately than did young children.

The computational bases for these effects were revealed by examining the pattern of connection weights within the CC network module. The pattern of these weights formed a cubic shape, symmetrical for the two sticks being compared, in which discrimination was better at the ends of the array than in the middle and became sharper with deeper learning.

Another task calls for integration of cues for moving objects, governed by the equation velocity = distance/time. Children were presented with information on two of those quantities and asked to infer the third. Three stages involved first using the quantity that varied positively with the quantity to be inferred, second adding or subtracting the known quantities, and finally multiplying or dividing the known quantities. Already documented stages were captured and others were correctly predicted by CC networks (Buckingham and Shultz 2000).

Semantic rules for deictic personal pronouns specify that *me* refers to the person using the pronoun and *you* refers to the person who is being addressed. Although most children acquire these pronouns without notable errors, a few reverse these pronouns, persistently calling themselves *you* and the mother *me*. Such reversals in children are produced by lack of opportunity to overhear these pronouns used by other people, where the shifting reference can be observed. CC networks covered these phenomena and generated predictions for effective therapy to correct reversal errors (Oshima-Takane et al. 1999).

Discrimination shift learning tasks repeatedly present pairs of stimuli with mutually exclusive attributes on several binary dimensions, such as color, shape, and position, and a child learns to select the correct stimulus in each pair, e.g., *square*. Feedback is given and learning continues until the child reaches a success criterion, e.g., 8/10 correct. Then reward contingencies shift, usually without warning. A *reversal* shift stays within the initially relevant dimension, e.g., from *square* to *circle*. A *nonreversal* shift is to another dimension, such as from *square* to *blue*. There are related tasks that use new stimulus values in the shift phase. These are called *intradimensional* shifts if the shift remains within the initial dimension, e.g., *square* to *triangle*, or *extradimensional* if there is a change to another dimension, e.g., from *square* to *yellow*. The *optional shift* task presents only two stimulus pairs in the shift phase, making it ambiguous whether the shift is a reversal or nonreversal shift. The pattern of subsequent choices allows determination of whether the child interprets this as a reversal or a nonreversal shift.

Age differences in the large literature on these shifts indicate that older children learn a reversal shift faster than a nonreversal shift, learn an intradimensional shift faster than an extradimensional shift, make a reversal shift in the optional task, and are initially impaired on unchanged pairs during a nonreversal shift. Younger children learn reversal and nonreversal shifts equally fast, learn an intradimensional shift faster than an extradimensional shift, make a nonreversal shift in the optional task, and are unimpaired on unchanged pairs during a nonreversal shift. These findings were simulated by CC networks (Sirois and Shultz 1998), which also generated predictions that were later confirmed.

When infants repeatedly experience stimuli from a particular class, their attention decreases, but it recovers to stimuli from a different class. This familiarize-and-test paradigm is responsible for most of the discoveries of infant psychological abilities. CC networks simulated findings on infant attention to syntactic patterns in an artificial language (Shultz and Bale 2006) and age differences in infant categorization of visual stimuli (Shultz and Cohen 2004) and generated several predictions, some of which were tested and confirmed.

SDCC

Because of SDCC's ability to create a variety of network topologies, it is beginning to be used in psychology simulations: infant learning of word-stress patterns in artificial languages (Shultz and Bale 2006), syllable boundaries (Shultz and Bale 2006), visual concepts (Shultz 2006), and false-belief tasks; learning the structure of mathematical groups (Schlimm and Shultz 2009); replication of the results of the CC simulation of conservation acquisition (Shultz 2006); and concept acquisition.

CC and SDCC networks capture developmental stages by growing in computational power and by being sensitive to statistical patterns in the training environment (Shultz 2003). The importance of growth was demonstrated by comparisons with static backprop networks, designed with the same final topology as successful CC networks, that learn only by adjusting connection weights (Shultz 2006). Coupled with the variety of successful SDCC topologies, this suggests that the constructive process is more important than precise network topologies. Capturing stages is challenging because the system has to not only succeed on the task but also make the same mistakes on the road to success that children do. CC and SDCC arguably produced the best data coverage of any models applied to the foregoing phenomena. Both static and constructive networks capture various perceptual effects by virtue of their sensitivity to quantitative variation in stimulus inputs (Shultz 2003).

Comparison of the two algorithms in psychological modeling indicates that SDCC provides the same functionality as CC but with fewer connection weights and shallower and more variable network topologies (Shultz 2006).

KBCC

KBCC also has potential for simulating psychological development, but it has so far been applied mainly to toy and engineering problems. Exploration of a variety of toy problems was important in understanding the behavior of this complex algorithm. Some toy problems involved learning about two-dimensional geometric shapes under various transformations such as translation, rotation, and size changes as well as compositions of complex shapes from simpler shapes (Shultz and Rivest 2001). Networks had to learn to distinguish points within a target shape from points outside the shape. Learning time without relevant knowledge was up to 16 times longer than with relevant knowledge on these problems. There was a strong tendency to recruit relevant knowledge whenever it was available. Direct comparison revealed that KBCC learned spatial translation problems faster than Multitask Learning networks did.

Parity problems require a network to activate an output unit only when an odd number of binary inputs are activated. When parity-4 networks were included in the candidate source pool, KBCC learned parity-8 problems (with eight binary inputs) faster and with fewer recruits than did CC networks. Parity-4 networks were recruited by these KBCC target networks whenever available.

KBCC also learned complex chessboard shapes from knowledge of simpler chessboards. As with parity, networks used simpler previous knowledge to compose a solution to a more complex problem and learning was speeded accordingly.

In a more realistic vein, KBCC networks recruiting knowledge of vowels from one sort of speaker (e.g., adult females) learned to recognize vowels spoken by other sets of speakers (e.g., children and adult males) faster than did knowledge-free networks.

KBCC learned an efficient algorithm for detecting prime numbers by recruiting previously learned knowledge of divisibility (Shultz et al. 2007). This well-known detection algorithm tests the primality of an integer n by checking if n is divisible by any integers between 2 and the integer part of $\sqrt{n}$. Starting with small primes is efficient because the smaller the prime divisor, the more composites are detected in a fixed range of integers. The candidate pool contained networks that had learned whether an integer could be divided by each of a range of integers, e.g., a divide- by-2 network, a divide-by-3 network, etc., up to a divisor of 20. KBCC target networks trained on 306 randomly selected integers from 21 to 360 recruited only source networks involving prime divisors below the square root of 360, in order from small to large divisors. KBCC avoided recruiting single hidden units, source networks with composite divisors, any divisors greater than the square root of 360 even if prime, and divisor networks with randomized connection weights. KBCC never recruited a divide-by-2 source network because it instead learned to check the last binary digit of n to determine if n was odd or even, an effective shortcut to dividing by 2. Such KBCC networks learned the training patterns in about one third the time required by knowledge-free networks, with fewer recruits on fewer network layers, and they generalized almost perfectly to novel test integers. In contrast, even after mastering the training patterns, CC networks generalized less well than automatic guessing that the integer was composite, which was true for 81 % of integers in this range. As predicted by the simulation, adults testing primality also used mainly prime divisors below $\sqrt{n}$ and ordered divisors from small to large.

This work underscores the possibility of neural network approaches to compositionality because KBCC effectively composed a solution to prime number detection by recruiting and organizing previously learned parts of the problem, in the form of divisibility networks.

Future Directions

One new trend is to inject symbolic rules or functions into KBCC source networks. This is similar to KBANN, but more flexible because a KBCC target network decides whether and how to recruit these functions. This provides one method of integrating symbolic and neural computation and allows for simulation of the effects of direct instruction.

Cross-References

- ▸ Artificial Neural Networks
- ▸ Backpropagation

Recommended Reading

Baluja S, Fahlman SE (1994) Reducing network depth in the cascade-correlation learning architecture. School of Computer Science, Carnegie Mellon University, Pittsburgh

Buckingham D, Shultz TR (2000) The developmental course of distance, time, and velocity concepts: a generative connectionist model. J Cognit Dev 1:305–345

Dandurand F, Berthiaume V, Shultz TR (2007) A systematic comparison of flat and standard cascade-correlation using a student-teacher network approximation task. Connect Sci 19:223–244

Fahlman SE (1988) Faster-learning variations on back-propagation: an empirical study. In: Touretzky DS, Hinton GE, Sejnowski TJ (eds) Proceedings of the 1988 connectionist models summer school. Morgan Kaufmann, Los Altos, pp 38–51

Fahlman SE (1991) The recurrent cascade-correlation architecture. In: Touretzky DS (ed) Advances in neural information processing systems, vol 3. Morgan Kaufmann, Los Altos

Fahlman SE, Lebiere C (1990) The cascade-correlation learning architecture. In: Touretzky DS (ed) Advances in neural information processing systems, vol 2. Morgan Kaufmann, Los Altos, pp 524–532

Mareschal D, Shultz TR (1999) Development of children's seriation: a connectionist approach. Connect Sci 11:149–186

Oshima-Takane Y, Takane Y, Shultz TR (1999) The learning of first and second pronouns in En-

glish: network models and analysis. J Child Lang 26:545–575
Schlimm D, Shultz TR (2009) Learning the structure of abstract groups. In: Taatgen NA, Rijn HV (eds) Proceedings of the 31st annual conference of the cognitive science society. Cognitive Science Society, Austin, pp 2950–2955
Shultz TR (1998) A computational analysis of conservation. Dev Sci 1:103–126
Shultz TR (2003) Computational developmental psychology. MIT, Cambridge
Shultz TR (2006) Constructive learning in the modeling of psychological development. In: Munakata Y, Johnson MH (eds) Processes of change in brain and cognitive development: attention and performance XXI. Oxford University Press, Oxford, pp 61–86
Shultz TR, Bale AC (2006) Neural networks discover a near-identity relation to distinguish simple syntactic forms. Minds Mach 16:107–139
Shultz TR, Cohen LB (2004) Modeling age differences in infant category learning. Infancy 5:153–171
Shultz TR, Mareschal D, Schmidt WC (1994) Modeling cognitive development on balance scale phenomena. Mach Learn 16:57–86
Shultz TR, Rivest F (2001) Knowledge-based cascade-correlation: using knowledge to speed learning. Connect Sci 13:1–30
Shultz TR, Rivest F, Egri L, Thivierge J-P, Dandurand F (2007) Could knowledge-based neural learning be useful in developmental robotics? The case of KBCC. Int J Humanoid Robot 4:245–279
Shultz TR, Takane Y (2007) Rule following and rule use in simulations of the balance-scale task. Cognition 103:460–472
Shultz TR, Vogel A (2004) A connectionist model of the development of transitivity. In: Proceedings of the twenty-sixth annual conference of the cognitive science society. Erlbaum, Mahwah, pp 1243–1248
Sirois S, Shultz TR (1998) Neural network modeling of developmental effects in discrimination shifts. J Exp Child Psychol 71:235–274

Cascor

▶ Cascade Correlation

Case

▶ Instance

Case-Based Learning

▶ Instance-Based Learning

Case-Based Reasoning

Susan Craw
Robert Gordon University, Aberdeen, UK

Abstract

Case-based reasoning (CBR) solves problems by retrieving similar, previously solved problems and reusing their solutions. The case base contains a set of cases, and each case holds knowledge about a problem or situation, together with its corresponding solution or action. The case base acts as a memory, remembering is achieved using similarity-based retrieval, and the retrieved solutions are reused. Newly solved problems may be retained in the case base and so the memory is able to grow as problem-solving occurs.

CBR reuses remembered experiences, where the experience need not record *how* the solution was reached, simply that the solution was used for the problem. The reliance on stored experiences means that CBR is particularly relevant in domains which are ill defined, not well understood, or where no underlying theory is available. CBR systems are a useful way to capture corporate memory of human expertise.

The fundamental assumption of CBR is that *similar problems have similar solutions*: a patient with similar symptoms will have the same diagnosis, the price of a house with similar accommodation and location will be similar, the design for a kitchen with a similar shape and size can be reused, and a journey plan is similar to an earlier trip. A related assumption is that the world is a regular place, and what holds true today will probably be true tomorrow. A further assumption relevant to memory is that situations repeat, because if they do not, there is no point remembering them!

Synonyms

Experience-based reasoning; Lessons-learned systems; Memory-based learning

Theory/Solution

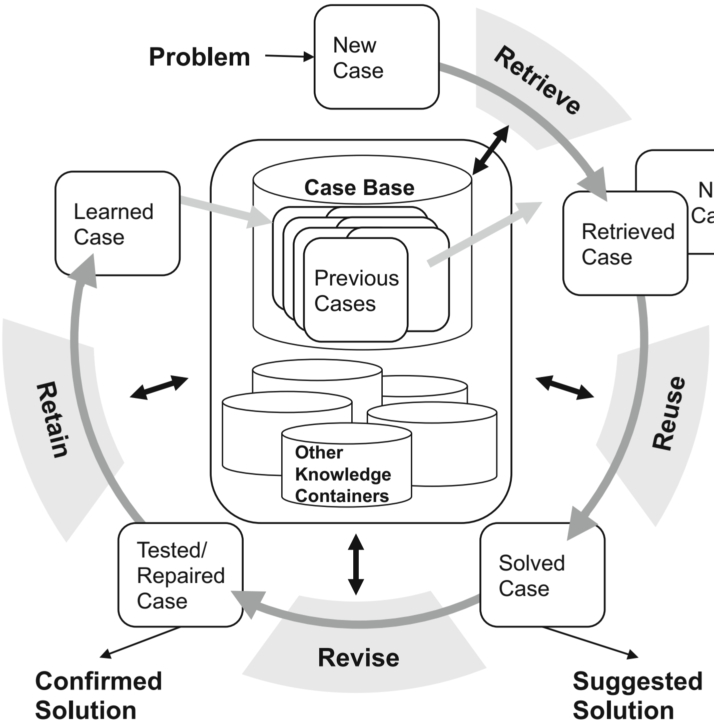
Case-based reasoning (CBR) is inspired by memory-based human problem-solving in which instances of earlier problem-solving are remembered and applied to solve new problems. For example, in case law, the decisions in trials are based on legal precedents from previous trials. In this way, specific experiences are memorized, and remembered and reused when appropriate. This contrasts with rule-based or theory-based problem-solving in which knowledge of *how* to solve a problem is applied. A doctor diagnosing a patient's symptoms may apply knowledge about how diseases manifest themselves, or she may remember a previous patient who demonstrated similar symptoms and thus apply a case-based approach.

CBR is an example of ▶ lazy learning because there is no learned model to apply to solve new problems. Instead, the generalization needed to solve unseen problems happens when a new problem is presented and the similarity-based retrieval identifies relevant previous experiences.

Figure 1 shows the CBR problem-solving cycle proposed by Aamodt and Plaza (1994). A case base of Previous Cases is the primary knowledge source in a CBR system, with additional knowledge being used to identify similar cases in the Retrieve stage, and to Reuse and Revise the retrieved case. A CBR system learns as it solves new problems when a Learned Case is created from the New Case and its Confirmed Solution, and Retained as a new case in the case base.

Aamodt and Plaza's four-stage CBR cycle for problem-solving and learning is commonly referred to as the "Four REs" or "R^4" cycle to recognize the following stages in Fig. 1:

- Retrieve: The cases that are most similar to the New Case defined by the description of the new problem are identified and retrieved from the case base. The Retrieve stage uses



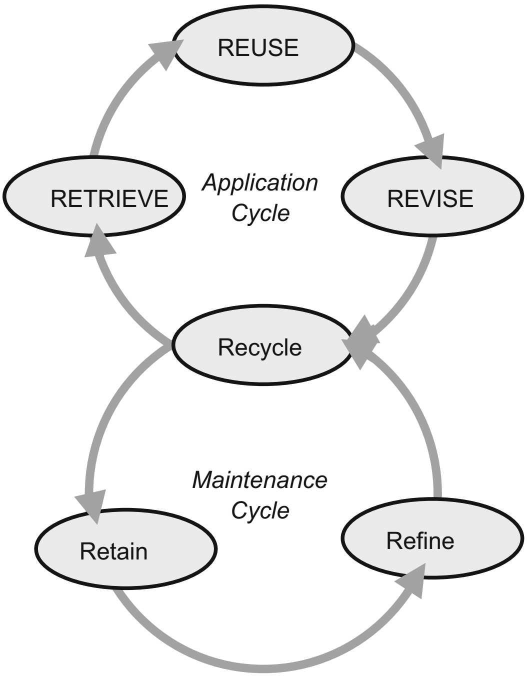
Case-Based Reasoning, Fig. 1 CBR cycle (Adapted from Aamodt and Plaza 1994) the similarity knowledge container in addition to the case base.

- Reuse: The solutions in the Retrieved (most similar) Cases are reused to build a Suggested Solution to create the Solved Case from the New Case. The Reuse stage may use the adaptation knowledge container to refine the retrieved solutions.
- Revise: The Suggested Solution in the Solved Case is evaluated for correctness and is repaired if necessary to provide the Confirmed Solution in the Tested/Repaired Case. The Revise stage may be achieved manually or may use adaptation knowledge, but it should be noted that a revision to a Suggested Solution is likely to be a less demanding task than solving the problem from scratch.
- Retain: The Repaired Case may be retained in the case base as a newly Learned Case if it is likely to be useful for future problem-solving. Thus the primary knowledge source for CBR may be added to during problem-solving and is an evolving, self-adaptive collection of problem-solving experiences.

This "Four REs" cycle simply Retained the Tested/Repaired Case as a new Learned Case. More recently, the Retain stage has been replaced with a Recycle-Retain-Refine loop of a "Six REs" cycle proposed by Gokër and Roth-Berghofer (1999) and shown in Fig. 2. Learned Cases are Recycled as potential new cases, the Retain step validates their correctness, before the Refine stage decides if the case should be integrated and how this should be done. The new case may be added, used to replace redundant cases, or merged with existing cases, and other case base maintenance may be required to maintain the integrity of the CBR system. The maintenance cycle is often executed separately from the problem-solving Application Cycle.

Knowledge Containers

Case knowledge is the primary source of knowledge in a CBR system. However, case knowledge is only one of four knowledge containers identified by Richter (2009):



Case-Based Reasoning, Fig. 2 Six REs CBR cycle (Adapted from Gokër and Roth-Berghofer 1999)

- Vocabulary: The representation language used to describe the cases captures the concepts involved in the problem-solving.
- Similarity knowledge: The similarity measure defines how the distances between cases are computed so that the nearest neighbors are identified for retrieval.
- Adaptation knowledge: Reusing solutions from retrieved cases may require some adaptation to enable them to fit the new problem.
- Case base: The stored cases capture the previous problem-solving experiences.

The content of each knowledge container is not fixed, and knowledge in one container can compensate for lack of knowledge in another. It is easy to see that a more sophisticated knowledge representation could be less demanding on the content of the case base. Similarly, vocabulary can make similarity assessment during retrieval easier, or a more complete case base could reduce the demands on adaptation during reuse. Further knowledge containers are proposed by others (e.g., maintenance by Gokër and Roth-Berghofer 1999) .

Cases may be represented as simple feature vectors containing nominal or numeric values. A case capturing a whisky-tasting experience might contain features such as sweetness, peatiness, color, nose and palate, and the ▶ classification as the distillery where it was made.

Sweet-ness	Peati-ness	Color	Nose	Palate	Distillery
6	5	amber	full	medium dry	Dalmore

More structured representations can use frame-based or object-oriented cases. The choice of representation depends on the complexity of the experiences being remembered and is influenced by how similarity should be determined. Hierarchical case representations allow cases to be remembered at different levels of abstraction, and retrieval and reuse may occur at these different levels.

For ▶ classification tasks, the case base can be considered to contain exemplars of problem-solving. This notion of exemplar confirms a CBR case base as a source of knowledge; it contains only those experiences that are believed to be useful for problem-solving. A similar view is taken for non-classification domains where the case base contains useful prototypes: for example, designs that can be used for redesign, plans for replanning, etc.

One of the advantages of CBR is that a case base is composed of independent cases that each captures some local problem-solving knowledge that has been experienced. Therefore, the "knowledge acquisition bottleneck" of many rule-based and model-based systems is reduced for CBR. However, the other knowledge containers provide additional knowledge acquisition demands that may lessen the advantage of CBR for some domains.

Retrieval

CBR retrieval compares the problem part of the new case with each of the cases in the case base to establish the distance between the new case and the stored cases. The cases closest to the new case are retrieved for reuse. Retrieval is a major focus of López de Mántaras et al.'s (2005) review of research contributions related to the CBR cycle.

Similarity- and distance-based neighborhoods are commonly used interchangeably when discussing CBR retrieval. Similarity and distance are inverses: the similarity is highest when the distance is close to 0, and the similarity is 0 when the distance is large. Several functions may be applied to define a suitable relationship between a distance d and a similarity s, including the following simple versions:

Inverse: $s = \dfrac{1}{d+1}$

Linear: $s = 1 - d$ for normalized d $(0 \leq d \leq 1)$

It is common to establish the distance between each pair of feature values and then to use a distance metric, often Euclidean or ▶ Manhattan distance (see also ▶ Similarity Measures), to calculate the distance between the feature vectors for the New and Retrieved Cases. The distance between two numeric feature values v and w for a feature F is normally taken to be the distance between the normalized values:

$$d(v, w) = \frac{\mid v - w \mid}{F_{\max} - F_{\min}}$$

where $F_{\max}/F_{\min}$ are the maximum/minimum values of the feature F.

For nominal values v and w, the simplest approach is to apply a binary distance function:

$$d(v, w) = \begin{cases} 0 \text{ if } v = w \\ 1 \text{ otherwise} \end{cases}$$

For ordered nominal values, a more fine-grained distance may be appropriate. The distance between the ith value v_i and the jth value v_j in the ordered values $v_1, v_2, \ldots, v_n$ may use the separation in the ordering to define the distance:

$$d(v_i, v_j) = \frac{\mid i - j \mid}{n - 1}.$$

Extending this to arbitrary nominal values, a distance matrix D may define the distance between each pair of nominal values by assigning the distance $d(v_i, v_j)$ to d_{ij}. Alternatively there may be background knowledge in the form of an ontology or concept hierarchy where their depth D in the structure compared to the depth of their least common ancestor (lca) can provide a measure of separation:

$$d(v_i, v_j) = \frac{D(v_i) + D(v_j)}{2D(\text{lca})}$$

Returning to the whisky-tasting example, suppose sweetness and peatiness score values 0–10, color takes ordered values {pale, straw, gold, honey, amber}, palate uses binary distance, and nose is defined by the following distance matrix:

Nose Distance Matrix

Distances	peat	fresh	soft	full
peat	0	0.3	1	0.5
fresh	0.3	0	0.5	0.7
soft	1	0.5	0	0.3
full	0.5	0.7	0.3	0

Dalmore whisky above can be compared with Laphroaig and The Macallan as follows:

Sweetness	Peatiness	Color	Nose	Palate	Distillery
2	10	amber	peat	medium dry	Laphroaig
7	4	gold	full	big body	The Macallan

The Manh distances are:

$$\begin{aligned} d(\text{Dalmore, Laphroaig}) &= 0.4 + 0.5 + 0 + 0.5 \\ &\quad + 0 = 1.4; \\ d(\text{Dalmore, The Macallan}) &= 0.1{+}0.1{+}0.5{+}0 \\ &\quad + 1 = 1.7. \end{aligned}$$

Taking all the whisky features with equal importance, Dalmore is more similar to Laphroaig than to The Macallan.

In situations where the relative importance of features should be taken into account, a weighted version of the distance function should be used; for example, the weighted Manhattan distance between two normalized vectors $\mathbf{x} = (x_1, x_2, \ldots x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots y_n)$ with weight w_i for the ith feature is

$$d(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^{n} w_i \mid x_i - y_i \mid}{\sum_{i=1}^{n} w_i}$$

In the example above, if Peatiness has weight 4 and the other features have weight 1, then the weighted Manhattan distances are:

$$\begin{aligned} d(\text{Dalmore, Laphroaig}) &= (0.4 + 4 \times 0.5 + 0 \\ &\quad +0.5{+}0)/8{=}0.36; \\ d(\text{Dalmore, The Macallan}) &= (0.1{+}4{\times}0.1{+}0.5 \\ &\quad +0 + 1)/8{=}0.25. \end{aligned}$$

Therefore, emphasizing the distinctive Peatiness feature, Dalmore is more similar to The Macallan than to Laphroaig.

The similarity knowledge container contains knowledge to calculate similarities. For simple feature vectors, a weighted sum of distances is often sufficient, and the weights are similarity knowledge. However, even our whisky-tasting domain had additional similarity knowledge containing the distance matrix for the nose feature. Structured cases require methods to calculate the similarity of two cases from the similarities of components. CBR may use very knowledge-intensive methods to decide similarity for the retrieval stage. Ease of reuse or revision may even be incorporated as part of the assessment of similarity. Similarity knowledge may also define how ▶ missing values are handled: the feature may be ignored, the similarity may be maximally pessimistic, or a default or average value may be used to calculate the distance.

A CBR case base may be indexed to avoid similarity matching being applied to all the cases in the case base. One approach uses kd trees to partition the case base according to hyperplanes. ▶ Decision Tree algorithms may be used to build the kd tree by using the cases as training data, partitioning the cases according to the chosen decision nodes and storing the cases in the appropriate leaf nodes. Retrieval first traverses the decision tree to select the cases in a leaf node, and similarity matching is applied to only this partition. Case Retrieval Nets are designed to speed up retrieval by applying spreading activation to select relevant cases. In Case Retrieval Nets, the feature value nodes are linked via similarity to each other and to cases. Indexes can speed up retrieval but they also preselect cases according to some criteria that may differ from similarity.

Reuse and Revision

Reuse may be as simple as copying the solution from the Retrieved Case. If k nearest neighbors are retrieved, then a vote of the classes predicted in the retrieved cases may be used for ▶ classification, or the average of retrieved values for ▶ regression. A weighted vote or weighted average of the retrieved solutions can take account of the nearness of the retrieved cases in the calculation. For more complex solutions, such as designs or plans, the amalgamation of the solutions from the Retrieved Cases may be more knowledge intensive.

If the New Case and the Retrieved Case are different in a significant way, then it may be that the solution from the Retrieved Case should be adapted before being proposed as a Suggested Solution. Adaptation is designed to recognize significant differences between the New and Retrieved Cases and to take account of these by adapting the solution in the Retrieved Case.

In classification domains, it is likely that all classes are represented in the case base. However, different problem features may alter the classification and so adaptation may correct for a lack of cases. In constructive problem-solving like design and planning, however, it is unlikely that all solutions (designs, plans, etc.) will be represented in the case base. Therefore, a retrieved case suggests an initial design or plan, and adaptation alters it to reflect novel feature values.

There are three main types of adaptation that may be used, as part of the reuse step to refine the solution in the Retrieved Case to match better the new problem, or as part of the revise stage to repair the Suggested Solution in the Solved Case:

- Substitution: Replace parts of the retrieved solution. In Hammond's (1990) CHEF system to plan Szechuan recipes, the substitution of ingredients enables the requirements of the new menu to be achieved. For example, the beef and broccoli in a retrieved recipe are substituted with chicken and snowpeas.
- Transformation: Add, change, or remove parts of the retrieved solution. CHEF adds a skinning step to the retrieved recipe that is needed for chicken but not for beef.
- Generative Adaptation: Replay the method used to derive the retrieved solution. Thus the retrieved solution is not adapted but a new solution is generated from reusing the retrieved method for the new circumstances. This approach is similar to reasoning by analogy.

CHEF also had a clear REVISE stage where the Suggested Solution recipe was tested in simulation and any faults were identified, explained, and repaired using repair templates for different types of explained failures. In one recipe a strawberry soufflé was too liquid, and one repair is to drain the strawberry pulp, and this transformation adaptation is one REVISE operation that could be applied.

The adaptation knowledge container is an important source of knowledge for some CBR systems, particularly for design and planning, where refining an initial design or plan is expected. Acquiring adaptation knowledge can be onerous, and learning adaptation knowledge from the cases in the case base or from background knowledge of the domain has been effective (Craw et al. 2006; Jalali and Leake 2013).

Retention and Maintenance

The retention of new cases during problem-solving is an important advantage of CBR systems. However, it is not always advantageous to retain all new cases. The ▶ Utility Problem – *that the computational benefit from additional knowledge must not outweigh the cost of applying* *it* – in CBR refers to cases and the added cost of retrieval. The case base must be kept "lean and mean," and so new cases are not retained automatically, and cases that are no longer useful are removed. New cases should be retained if they add to the competence of the CBR system by providing problem-solving capability in an area of the problem space that is currently sparse. Conversely, existing cases should be reviewed for the role they play, and forgetting cases is an important maintenance task. Existing cases may contain outdated experiences and so should be removed, or they may be superseded by new cases.

Case base maintenance manages the contents of the case base to achieve high competence. Competence depends on the domain and may involve:

- quality of solution;
- user confidence in solution; or
- efficiency of solution prediction (e.g., speed-up learning).

As a result, the RETAIN step in Aamodt and Plaza's (1994) "four REs" problem-solving cycle is normally replaced by some form of case base maintenance cycle, such as the ReCycle-Retain-Refine loop in Gokër and Roth-Berghofer's (1999) "six REs" cycle.

Case base maintenance systems commonly assume that the case base contains a representative sample of the problem-solving experiences. They exploit this by using a leave-one-out approach where repeatedly for each case in the case base, the one extracted case is used as a new case to be solved, and the remaining cases become the case base. This enables the problem-solving competence of the cases in the case base to be estimated using the extracted cases as representative new cases to be solved. Various researchers build a competence model for the case base by identifying groups of cases with similar problem-solving ability and use this model to underpin maintenance algorithms that prioritize cases for deletion and to identify areas where new cases might be added.

There are several trade-offs to be managed by case base maintenance algorithms: larger case bases contain more experiences but take longer for retrieval; smaller case bases are likely to lack some key problem-solving ability; cases whose solution is markedly different from their nearest neighbors may be noisy or may be an important outlier. The competence of a case depends on other knowledge containers, and so case base maintenance should not proceed in isolation.

CBR Applications and Tools

Two notable successful deployed applications of CBR are Verdande's Drilledge that monitors oil-well drilling operations to reduce non-productive time (Gundersen et al. 2013), and General Electric's FormTool for plastic color matching (Cheetham 2005). Many more applications are described in the *Fielded Applications of CBR* article in Knowledge Engineering Review 20(3) CBR Special Issue (2005) and Montani and Jain's *Successful Case-Based Reasoning Applications* texts (Springer, 2010 & 2014):

- *Classification* – Medical diagnosis systems include SHRINK for psychiatry, CASEY for cardiac disease, and ICONS for antibiotic therapy for intensive care. Other diagnostic systems include failure prediction of rails for Dutch railways, Boeing's CASSIOPÉE for trouble-shooting aircraft engines, and the HOMER Help-Desk (Gokër and Roth-Berghofer 1999).
- *Design* – Architectural design was a popular early domain: ARCHIE and CADsyn. Other design applications include CADET and KRITIK for engineering design, pharmaceutical tablet formulation, Déjà Vu for plant control software, and Lockheed's CLAVIER for designing layouts for autoclave ovens.
- *Planning* – PRODIGY is a general purpose planner that uses analogical reasoning to adapt retrieved plans. Other planning applications include PARIS for manufacturing planning, mission planning for US navy, and route planning for DaimlerChrysler cars. A recent focus is planning in simulated complex environments as found in Real-Time Strategy Games (Jaidee et al. 2013; Ontañón and Ram

2011; Wender and Watson 2014). CBR has other Game AI applications including robot soccer and poker.

- *Textual CBR* – Legal decision support systems were an important early application domain for textual CBR, including HYPO, GREBE, and SMILE. Question answering was another fruitful text-based domain: FAQ-Finder and FA11Q. More recently, textual CBR is used for industrial decision support based on textual reports; e.g., incident management and Health & Safety.
- *Conversational CBR* – Conversational systems extract the problem specification from the user through an interactive case-based dialogue. Examples include help-desk support, CBR Strategist for fault diagnosis, and Wasabi and ShowMe product recommender systems.
- *Recommender Systems* – There has been a large growth in the use of CBR for recommendation of products, travel planning, and online music. Current topics include preference recommenders for individuals and groups (Quijano-Sánchez et al. 2012) and sentiment/opinion mining from social media to improve personalization (Dong et al. 2014).
- *Workflows* – A recent interest in process-oriented CBR has used the CAKE Collaborative Agile Knowledge Engine to create office workflows (Minor et al. 2014). Other applications include science workflows, medical pathways, modeling interaction traces, and recipes. These applications use structured cases and demand knowledge-rich adaptation for reuse. An annual Computer Cooking Competition at recent ICCBR conferences has encouraged the development of various case-based recipe systems including Taaable, JADAWeb, CookIIS, ChefFroglingo, GoetheShaker (cocktails), and EARL (sandwiches).

There are two main open-source CBR tools: myCBR and Colibri. Both provide state-of-the-art CBR functionality, and Colibri also incorporates a range of facilities for textual CBR. The myCBR tool originated from the INRECA methodology, and its website www.mycbr-project.net offers downloads, documentation, tutorials, and publications. Similar Colibri information is available at gaia.fdi.ucm.es/research/colibri, with the jColibri framework also available from www.sourceforge.net. Empolis is one of the pioneers in CBR with CBR Works being one of the first commercial CBR tools. It is now part of Empolis' Information Access System, and is available at www.empolis.com.

Future Directions

The drivers for ubiquitous computing – wireless communication and small devices – also affect future developments in CBR. The local, independent knowledge of case bases makes mobile devices ideal to collect experiences and to deliver experience-based knowledge for reuse.

Textual CBR systems are becoming increasingly important for extracting and representing knowledge captured in textual documents. This is particularly influenced by the availability of electronic documents in the Web and social media as sources of data for the extraction of representation knowledge. They also provide background knowledge from which to learn knowledge for similarity and adaptation containers.

Cross-References

▶ Instance-Based Learning
▶ Lazy Learning
▶ Nearest Neighbor
▶ Similarity Measures

Recommended Reading

Aamodt A, and Plaza E (1994) Case-based reasoning: foundational issues, methodological variations, and system approaches. AI Commun 7:39–59. citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.1670

Cheetham W (2005) Tenth anniversary of the plastics color formulation tool. AI Mag 26(3):51–61 www.aaai.org/Papers/Magazine/Vol26/26-03/AIMag26-03-007.pdf

Craw S, Wiratunga N, Rowe RC (2006) Learning adaptation knowledge to improve case-based reasoning. Artif Intell 170(16–17):1175–1192. doi:10.1016/j.artint.2006.09.001

Dong R, Schaal M, O'Mahony MP, McCarthy K, Smyth B (2014) Further experiments in opinionated product recommendation. In: Lamontagne L, Plaza E (eds) Proceedings of the 22nd international conference on case-based reasoning, Cork. LNAI, vol 8765. Springer, Berlin/Heidelberg, pp 110–124. doi:10.1007/978-3-319-11209-1_9

Gokër MH, Roth-Berghofer T (1999) The development and utilization of the case-based help-desk support system HOMER. Eng Appl Artif Intell 12:665–680. doi:10.1016/S0952-1976(99)00037-8

Gundersen OE, Sørmo F, Aamodt A, Skalle P (2013) A real-time decision support system for high cost oil-well drilling operations. AAAI AI Mag 34(1): 21–31. www.aaai.org/ojs/index.php/aimagazine/article/view/2434

Hammond KJ (1990) Explaining and repairing plans that fail. Artif Intell 45(1–2):173–228

Jaidee U, Muñoz-Avila H, Aha DW (2013) Case-based goal-driven coordination of multiple learning agents. In: Delaney SJ, Ontanon S (eds) Proceedings of the 21st international conference on case-based reasoning, Saratoga Springs. LNAI, vol 7969. Springer, Berlin/Heidelberg, pp 164–178. doi:10.1007/978-3-642-39056-2_12

Jalali V, Leake D (2013) Extending case adaptation with automatically-generated ensembles of adaptation rules. In: Delaney SJ, Ontanon S (eds) Proceedings of the 21st international conference on case-based reasoning, Saratoga Springs. LNAI, vol 7969. Springer, Berlin/Heidelberg, pp 188–202. doi:10.1007/978-3-642-39056-2_14

López de Mántaras R, McSherry D, Bridge D, Leake D, Smyth B, Craw S, Faltings B, Maher ML, Cox MT, Forbus K, Aamodt A, Watson I (2005) Retrieval, reuse, revision, and retention in case-based reasoning. Knowl Eng Rev 20(3):215–240. doi:10.1017/S0269888906000646

Minor M, Bergmann R, Görg S (2014) Case-based adaptation of workflows. Inf Syst 40:142–152. doi:10.1016/j.is.2012.11.011

Ontañón S, Ram A (2011) Case-based reasoning and user-generated AI for real-time strategy games. In: Artificial intelligence for computer games. Springer, New York, pp 103–124. doi:10.1007/978-1-4419-8188-2_5

Quijano-Sánchez L, Bridge D, Díaz-Agudo B, Recio-García JA (2012) Case-based aggregation of preferences for group recommenders. In: Díaz-Agudo B, Watson I (eds) Proceedings of the 20th international conference on case-based reasoning, Lyon. LNAI, vol 7466. Springer, Berlin/Heidelberg, pp 17–31. doi:10.1007/978-3-642-32986-9_25

Richter MM (2009) The search for knowledge, contexts, and case-based reasoning. Eng Appl Artif Intell 22(1):3–9. doi:10.1016/j.engappai.2008.04.021

Wender S, Watson I (2014) Combining case-based reasoning and reinforcement learning for unit navigation in real-time strategy game AI. In: Lamontagne L, Plaza E (eds) Proceedings of the 22nd international conference on case-based reasoning, Cork. LNAI, vol 8765. Springer, Berlin/Heidelberg, pp 511–525. doi:10.1007/978-3-319-11209-1_36

Categorical Attribute

Synonyms

Qualitative attribute

Categorical attributes are attributes whose values can be placed into distinct categories See ▶ Attribute and ▶ Measurement Scales.

Categorical Data Clustering

Periklis Andritsos[1] and Panayiotis Tsaparas[2]
[1]Faculty of Information, University of Toronto, Toronto, ON, Canada
[2]Department of Computer Science & Engineering, University of Ioannina, Ioannina, Greece

Abstract

In this chapter, we provide an overview of the categorical data clustering problem. We first present different techniques for the general cluster analysis problem, and then study how these techniques specialize to the case of non-numerical (categorical) data. We also present measures and techniques developed specifically for this domain.

Synonyms

Clustering of nonnumerical data; Grouping

Definition

Data clustering is informally defined as the problem of partitioning a set of objects into groups, such that objects in the same group are similar, while objects in different groups are dissimilar. Categorical data clustering refers to the case where the data objects are defined over *categorical* attributes. A categorical attribute is an attribute whose domain is a set of discrete values that are not inherently comparable. That is, there is no single ordering or inherent distance function for the categorical values, and there is no mapping from categorical to numerical values that is semantically sensible.

Motivation and Background

Clustering is a problem of great practical importance that has been the focus of substantial research in several domains for decades. As the volume of collected data grows, the need to mine and understand the data becomes imperative. Clustering plays an instrumental role in this process. As a result in the recent years, there has been a surge of research activity in devising new clustering algorithms that can handle large amounts of data and produce results of high quality.

In data clustering we want to partition objects into groups such that similar objects are grouped together, while dissimilar objects are grouped separately. This definition assumes that there is some well-defined notion of similarity, or distance, between data objects, and a way to decide if a group of objects is a homogeneous cluster. Most of the clustering algorithms in the literature focus on data sets where objects are defined over numerical attributes. In such cases, the similarity (or dissimilarity) of objects can be defined using well-studied measures that are derived from geometric analogies. These definitions rely on the semantics of the data values themselves (e.g., the values $100,000 and $110,000 are more similar than $100,000 and $1). The existence of a distance measure enables us to define a quality measure for a clustering (e.g., the mean square distance between each point and the representative of its cluster). Clustering then becomes the problem of grouping together points such that the quality measure is optimized.

Categorical Data Clustering, Table 1 An instance of a movie database

	Director	Actor	Genre
$\mathbf{t}_1$ (Godfather II)	Scorsese	De Niro	Crime
$\mathbf{t}_2$ (Good fellas)	Coppola	De Niro	Crime
$\mathbf{t}_3$ (Vertigo)	Hitchcock	Stewart	Thriller
$\mathbf{t}_4$ (N by NW)	Hitchcock	Grant	Thriller
$\mathbf{t}_5$ (Bishop's wife)	Koster	Grant	Comedy
$\mathbf{t}_6$ (Harvey)	Koster	Stewart	Comedy

However, there are many data sets where the data objects are defined over attributes, which are neither numerical nor inherently comparable in any way. We term such data sets *categorical*, since they represent values of certain categories. As a concrete example, consider the toy data set in Table 1 that stores information about movies. For the purpose of exposition, a movie is characterized by the attributes "director," "actor/actress," and "genre." In this setting, it is not immediately obvious what the distance, or similarity, is between the values "Coppola" and "Scorsese" or the tuples "Vertigo" and "Harvey."

There are plenty of examples of categorical data: product data, where products are defined over attributes such as brand, model, or color; census data, where information about individuals includes attributes such as marital status, address, and occupation; ecological data where plants and animals can be described with attributes such as shape of petals or type of habitat. There is a plethora of such data sets, and there is always a need for clustering and analyzing them.

The lack of an inherent distance or similarity measure between categorical data objects makes clustering of categorical data a challenging problem. The challenge lies in defining a quality measure for categorical data clustering that captures the human intuition of what it means for categorical data objects to be similar. In the following, we will present an overview of the different efforts at addressing this problem and the resulting clustering algorithms.

Structure of the Learning System

Generic Data Clustering System

We first describe the outline for a generic data clustering system, not necessarily of categorical data. We will then focus on techniques specific to categorical data clustering.

Data clustering is not a one-step process. In one of the seminal texts on cluster analysis, Jain and Dubes (1988) divided the clustering process into seven different stages starting from the data collection and ending with the result interpretation. In this article we are interested in problems relating to the representation of the data, which affects the notion of similarity between the categorical tuples and the clustering strategy, which determines the final algorithm. These lie in the heart of the data clustering problem, and there has been considerable research effort in these areas within the data mining and machine learning communities. More specifically we will consider the following two subproblems: (a) the formal formulation of the clustering problem and (b) the clustering algorithm.

Formal formulation of the clustering problem: In order to devise algorithms for clustering, we need to mathematically formulate the intuition behind the informal definition of the clustering problem where "similar objects are grouped together and dissimilar objects are grouped separately." The problem formulation typically requires at least one of the following:

- A measure of similarity or distance between two data objects.
- A measure of similarity or distance between a data object and a cluster of objects. This is often defined by defining a representative for a cluster as a (new) data object and comparing the data object with the representative.
- A measure of the quality of a cluster of data objects.

The result of the formulation step is to define a clustering optimization criterion that guides the grouping of the objects into clusters.

When the data is defined over numerical attributes, these measures are defined using geometric analogies. For example, in one possible formulation, we can view each object as a point in the Euclidean space, and define the distance between two points as the Euclidean distance, and the representative of a cluster as the mean Euclidean vector. The quality of a cluster can be defined with respect to the "variance" of the cluster, that is, the sum of squares of the distances between each object and the mean of the cluster. The optimization problem then becomes to minimize the variance over all clusters.

The clustering algorithm: Once we have a mathematical formulation of the clustering problem, we need an algorithm that will find the optimal clustering in an efficient manner. In most cases finding the optimal solution is an NP-hard problem, so there are a variety of efficient heuristics or approximation algorithms. There is an extensive literature on this subject that approaches the problem from different angles and a wide variety of different clustering techniques and algorithms. We now selectively describe some broad classes of clustering algorithms and problems. A thorough categorization of clustering techniques can be found in Han and Kamber (2001), where different clustering problems, paradigms, and techniques are discussed.

Hierarchical clustering algorithms: This is a popular clustering technique since it is easy to implement, and it lends itself well to visualization and interpretation. Hierarchical algorithms create a hierarchical decomposition of the objects. They are either *agglomerative* (*bottom-up*) or *divisive* (*top-down*). *Agglomerative* algorithms start with each object being a separate cluster itself and successively merge groups according to some criterion. *Divisive* algorithms follow the opposite strategy. They start with one cluster consisting of all objects and successively split clusters into smaller ones, until each object falls in one cluster, or the desired given condition is met. The hierarchical *dendrogram* produced is often in itself the output of the algorithm, since it can be used for visualizing the data. Most of the times, both approaches suffer from the fact that once a merge or a split is committed, it cannot be undone or refined.

Partitional clustering algorithms: Partitional clustering algorithms define a clustering optimization criterion and then seek the partition that optimizes this criterion. Exhaustive search over all partitions is infeasible since even for few data objects, the number of possible partitions is prohibitively large. Partitional clustering algorithms often start with an initial, usually random, partition and proceed with its refinement by locally improving the optimization criterion. The majority of them could be considered as greedy-like algorithms. They suffer from the fact that they can easily get stuck to local optima.

Spectral clustering algorithms: Spectral algorithms use the data set to be clustered to construct a two-dimensional matrix of data objects and attributes. The entries in the matrix may be the raw values or some transformation of these values. The principal eigenvectors of the matrix are then used to reveal the clustering structure in the data. There is a rich literature on different types of spectral algorithms.

Graph clustering: Graph clustering defines a range of clustering problems, where the distinctive characteristic is that the input data is represented as a graph. The nodes of the graph are the data objects, and the (possibly weighted) edges capture the similarity or distance between the data objects. The data may come naturally in the form of a graph (e.g., a social network), or the graph may be derived in some way from the data (e.g., link two products if they appear together in a transaction). Some of the techniques we describe above are directly applicable to graph data.

Categorical Data Clustering System

In the clustering paradigm we outlined, a step of fundamental importance is to formally formulate the clustering problem, by defining a clustering optimization criterion. As we detail above, for this step we need a measure of distance or similarity between the data objects or a measure of cluster quality for a group of data objects. For categorical data there exists no inherent ordering or distance measure, and no natural geometric analogies we can explore, causing the clustering paradigm to break down. Research efforts on categorical data clustering have focused on addressing this problem by imposing distance measures on the categorical data and defining clustering quality criteria. We now outline some of these approaches.

Overlap-based similarity measures: A simple and intuitive method for comparing two categorical data objects is to view them as sets of attribute values and count the overlap between the categorical values of the objects. The higher the overlap, the more similar the two objects are. This intuition leads to the use of well-known measures such as the (*generalized*) *Hamming distance* (Jain and Dubes 1988), which measures the number of common values between two tuples, or the *Jaccard* similarity measure, which is defined as the intersection over the union of the values in the two tuples. In the example of Table 1, tuples $\mathbf{t}_1$ and $\mathbf{t}_2$ have Hamming distance 1 and Jaccard coefficient $1/2$.

Two algorithms that make use of overlap-based measures are *k-modes* (Huang 1998) and *ROCK (RObust Clustering using linKs)* (Guha et al. 1999). The k-modes algorithm is a partitional algorithm inspired by the *k-means* algorithm, a well-known clustering algorithm for numerical data. The representative of a categorical data cluster is defined to be a data object where each attribute takes the *mode* value: the mode of an attribute is the most frequent attribute value in the cluster. The ROCK algorithm makes use of the Jaccard coefficient to define *links* between data objects. The data is then represented in the form of a graph, and the problem becomes essentially a graph clustering problem. Given two clusters of categorical data, ROCK measures the similarity of two clusters by comparing their *aggregate interconnectivity* against a user-specified model, thus avoiding the problem of defining a cluster representative.

Context-based similarity measures: One way to define relationships between categorical values is by comparing the *context* in which they appear. For two categorical attribute values, we define the context as the values of other attributes with which they co-occur in the data set. The more similar these two contexts are, the more similar the attribute values are. For example, in Table 1, Scorsese and Coppola are close since they appear in exactly the same context, while Scorsese and Hitchcock are far since their contexts are completely disjoint. Defining a distance between value contexts can be done using overlap similarity measures (Das and Mannila 2000) or by using information-theoretic measures, i.e., comparing the distributions defined by the two contexts (Andritsos et al. 2004). Once we have the relationships between the values, we can use standard clustering techniques for solving the clustering problem.

There are various algorithms that make use of the idea that similar values should appear in similar contexts in order to cluster categorical data. The *CACTUS (clustering categorical data using summaries)* algorithm (Ganti et al. 1999) creates groups of attribute values based on the similarity of their context. It then uses a hierarchical greedy algorithm for grouping tuples and attributes. In a slightly different fashion, *STIRR (sieving through iterated relational reinforcement)* (Ganti et al. 1998) uses the idea that similar tuples should contain co-occurring values, and similar values should appear in tuples with high overlap. This idea is implemented via a dynamical system, inspired by information retrieval techniques. When the dynamical system is linear, the algorithm is similar to spectral clustering algorithms. *CLICKS* (Zaki et al. 2005) is an algorithm that is similar to STIRR. Rather than a measure of similarity/distance, it uses a graph-theoretic approach to find k disjoint sets of vertices in a graph constructed for a particular data set. One special characteristic of this algorithm is that it discovers clusters in a subset of the underlying set of attributes.

Information-theoretic clustering criteria: The information content in a data set can be quantified through the well-studied notions of *entropy* and *mutual information* (Cover and Thomas 1991). Entropy measures the uncertainty in predicting the values of the data when drawn from a distribution. If we view each tuple, or cluster of tuples, as a distribution over the categorical values, then we can define the *conditional entropy* of the attribute values given a set of tuples, as the uncertainty of predicting the values in this set of tuples. If we have a single tuple, then the entropy is zero, since we can accurately predict the values. For tuple $\mathbf{t}_1$ we know the director, the actor, and the genre with full certainty. As we group tuples together, the uncertainty (and entropy) increases. Grouping together tuples $\mathbf{t}_1$ and $\mathbf{t}_2$ creates uncertainty about the director attribute, while grouping $\mathbf{t}_1$ and $\mathbf{t}_3$ creates uncertainty about all attributes. Hence the latter grouping has higher entropy than the former. Information-theoretic criteria for clustering aim at generating clusters with low entropy, since this would imply that the clusters are homogeneous, and there is little *information loss*. This formulation allows for defining the distance between sets of tuples, using entropy-based divergences, such as the *Jensen-Shannon* divergence (Cover and Thomas 1991). Jensen-Shannon divergence captures the information contained in the data set, in a similar way that mean-squared distance captures geometric notions inherent in numerical data.

Two algorithms that make use of this idea are *COOLCAT* (Barbarà et al. 2002) and *LIMBO (scalable information bottleneck)* (Andritsos et al. 2004). COOLCAT is a partitional algorithm that performs a local search for finding the partition with k clusters with the lowest entropy. LIMBO works by constructing a summary of the data set that preserves as much information about the data as possible and then produce a hierarchical clustering of the summary. It is a scalable algorithm that can be used in both static and streaming environments.

A related approach is adopted by the COBWEB algorithm (Fisher 1987; Gluck and Corter 1985), a divisive hierarchical algorithm that optimizes the *category utility* measure, which measures how well particular values can be predicted given the clustering as opposed to having them in the original data set unclustered.

Categorical clustering as clustering aggregation: A different approach to the categorical data clustering problem is to view it as a *clustering aggregation problem*. Given a collection of clusterings of the data objects, the clustering aggregation problem looks for the single clustering that agrees as much as possible from the input clusterings. The problem of clustering aggregation has been shown to be equivalent to categorical data clustering (Gionis et al. 2007), where each categorical attribute defines a clustering of the data objects, grouping all the objects with the same value together. For example, in Table 1, the attribute "genre" defines three clusters: the crime cluster, the thriller cluster, and the comedy cluster. Similarly, the attribute "actor" defines three clusters, and the attribute "director" defines four clusters.

Various definitions have been considered in the literature for the notion of agreement between the output clustering and the input clusterings. One definition looks at all pairs of objects and defines a *disagreement* between two clusterings if one clustering places the two objects in the same cluster, while the other places them in different clusters; an *agreement* is defined otherwise. The clustering criterion is then to minimize the number of disagreements (or maximize the number of agreements). Other definitions are also possible, which make use of information-theoretic measures or mappings between the clusters of the two clusterings. There is a variety of algorithms for finding the best aggregate cluster, many of which have also been studied theoretically.

Cross-References

- ▶ Data mining on Text
- ▶ Instance-Based Learning

Recommended Reading

Andritsos P, Tsaparas P, Miller RJ, Sevcik KC (2004) LIMBO: scalable clustering of categorical data. In: Proceedings of the 9th international conference on extending database technology (EDBT), Heraklion, 14–18 Mar 2004, pp 123–146

Barbarà D, Couto J, Li Y (2002) COOLCAT: an entropy-based algorithm for categorical clustering. In: Proceedings of the 11th international conference on information and knowledge management (CIKM), McLean, 4–9 Nov 2002, pp 582–589

Cover TM, Thomas JA (1991) Elements of information theory. Wiley, New York

Das G, Mannila H (2000) Context-based similarity measures for categorical databases. In: Proceedings of the 4th European conference on principles of data mining and knowledge discovery (PKDD), Lyon, 13–16 Sept 2000, pp 201–210

Fisher DH (1987) Knowledge acquisition via incremental conceptual clustering. Mach Learn 2: 139–172

Ganti V, Gehrke J, Ramakrishnan R (1999) CACTUS: clustering categorical data using summaries. In: Proceedings of the 5th international conference on knowledge discovery and data mining, (KDD), San Diego, 15–18 Aug 1999, pp 73–83

Gionis A, Mannila H, Tsaparas P (2007) Clustering aggregation. In: ACM transactions on knowledge discovery from data (TKDD), Mar 2007, vol 1, No 1. Association for Computing Machinery, New York

Gibson D, Kleinberg JM, Raghavan P (1998) Clustering categorical data: an approach based on dynamical systems. In: Proceedings of the 24rth international conference on very large data bases, (VLDB), New York, 24–27 Aug 1998, pp 311–322

Gluck M, Corter J (1985) Information, uncertainty, and the utility of categories. In: Proceedings of the 7th annual conference of the Cognitive Science Society (COGSCI), Irvine, pp 283–287

Guha S, Rastogi R, Shim K (1999) ROCK: a robust clustering algorithm for categorical atributes. In: Proceedings of the 15th international conference on data engineering, Sydney, 23–26 Mar 1999, pp 512–521

Jain AK, Dubes RC (1988) Algorithms for clustering data. Prentice-Hall, Englewood Cliffs

Jarke M, Lenzerini M, Vassiliou Y, Vassiliadis P (1999) Fundamentals of data warehouses. Springer-Verlag, Berlin/Heidelberg

Han J, Kamber M (2001) Data mining: concepts and techniques. Morgan Kaufmann, San Francisco

Huang Z (1998) Extensions to the k-means algorithm for clustering large data sets with categorical values. Data Min Knowl Discov 2(3):283–304

Zaki MJ, Peters M, Assent I, Seidl T (2005) CLICKS: an effective algorithm for mining subspace clusters in categorical datasets. In: Proceeding of the 11th international conference on knowledge discovery and data mining (KDD), Chicago, 21–24 Aug 2005, pp 736–742

Categorization

▶ Classification
▶ Concept Learning

Category

▶ Class

Causal Discovery

▶ Learning Graphical Models

Causality

Ricardo Silva
Centre for Computational Statistics and Machine Learning, University College London, London, UK

Abstract

Causality is an essential concept in our understanding of the world as, in order to predict how a system behaves under an intervention, it is necessary to have causal knowledge of the impact of interventions. This knowledge should be expressed in a language built on top of probabilistic models, since the axioms of probability do not provide a way of expressing how external interventions affect a system. Learning this knowledge from data also poses additional challenges compared to the standard machine learning problem, as much data comes from passive observations that do not follow the same regime under which our predictions will take place.

Definition

The main task in causal inference is the prediction of the outcome of an intervention. For example, a treatment assigned by a doctor that will change the patient's heart condition is an intervention. Predicting the change in patient condition is a causal inference task. In general, an intervention is an action taken by an external agent that changes the original values, or the probability distributions, of some of the variables in the system. Besides predicting outcomes of actions, causal inference is also concerned with explanation: identifying which were the causes of a particular event that happened in the past.

Motivation and Background

Many problems in machine learning are prediction problems. Given a feature vector $\mathbf{X}$, the task is to provide an estimate of some output vector $\mathbf{Y}$ or its conditional probability distribution $P(\mathbf{Y}|\mathbf{X})$. This typically assumes that the distribution of $\mathbf{Y}$ given $\mathbf{X}$ during learning is the same distribution at prediction time. There are many scenarios where this is not the case.

Epidemiology and several medical sciences provide counterexamples. Consider two seemingly straightforward learning problems. In the first example, one is given epidemiological data where smokers are clearly more inclined than nonsmokers to develop lung cancer. Can I use this data to learn that smoking causes cancer? In the second example, consider a group of patients suffering from a type of artery disease. In this group, those that receive a bypass surgery are likely to survive longer than those that receive a particular set of drugs with no surgery.

There is no fundamental problem on using such datasets to predict the probability of a smoker developing lung cancer or the life expectancy of someone who went through surgery. Yet, the data does not necessarily tell you if smoking is a cause of lung cancer or that nationwide the government should promote surgery as the treatment of choice for that particular heart disease. What is going on?

There are reasons to be initially suspicious of such claims. This is well known in statistics as the expression "association is not causation" (Wasserman 2004, p. 253). The data generating mechanism for our outcome **Y** ("developing lung cancer," "getting cured from artery disease") given the relevant inputs **X** ("smoking habit," "having a surgery") might change under an *intervention* for reasons such as follows.

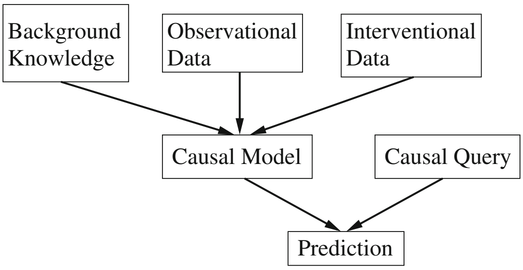
In the smoking example, the reality might be that there are several *hidden common causes* that are responsible for the observed association. A genetic factor includes, for instance, the possibility that there is a class of genotypes on which people are more likely to pick up smoking *and* develop lung cancer, without any direct causal connection between these two variables. In the artery disease example, surgery might not be the best choice to be made by a doctor. It might have been the case that so far patients in better shape were more daring in choosing, by themselves, the surgery treatment. This *selection bias* will favor surgery over drug treatment, since from the outset the patients that are most likely to improve take that treatment.

When treatment is enforced by an *external agent* (the doctor), such selection bias disappears, and the resulting $P(\mathbf{Y}|\mathbf{X})$ will not be the same. One way of learning this relationship is through *randomized trials* (Rosenbaum 2002). The simplest case consists on flipping a coin for each patient on the training set. Each face of the coin corresponds to a possible treatment, and assignment is done accordingly. Since assignment does not depend on any hidden common cause or selection bias, this provides a basis for learning causal effects. Machine learning and statistical techniques can be applied directly in this case (e.g., logistic regression). Data analysis performed with a randomized trial is sometimes called an *interventional study*.

The smoking case is more complicated: a direct intervention is not possible, since it is not acceptable to force someone to smoke or not to smoke. The inquiry asks only for a *hypothetical intervention*, i.e., if someone is forced to smoke, will his or her chances of developing lung cancer increase? Such an intervention will not take place, but this still has obvious implications in public policy. This is the heart of the matter in issues such as deciding on raising tobacco taxes or forbidding smoking in public places. However, data that measures this interventional data generation mechanism will never be available for ethical reasons. The question has to be addressed through an *observational study*, i.e., a study for causal predictions without interventional data.

Observational studies arise not only under the impossibility of performing interventions but also in the case where performing interventions is too expensive or time consuming. In this case, observational studies, or a combination of observational and interventional studies, can provide extra information to guide an experimental analysis (Hyttinen et al. 2013; Sachs et al. 2005; Cooper and Yoo 1999; Eaton and Murphy 2007). The use of observational data, or the combination of several interventional datasets, is where the greatest contributions of machine learning to causal inference rest.

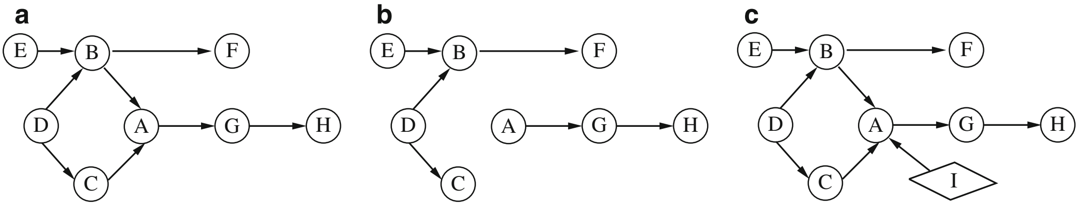
Structure of the Learning System



Structure of Causal Inference

In order to use observational data, a causal inference system needs a way of linking the state of the world under an intervention to the *natural* state of the world. The natural state is defined as the one to which no external intervention is applied. In the most general formulation, this link between the natural state and the manipulated world is defined for interventions in any subset of variables in the system.

A common language for expressing the relationship between the different states of the world is a *causal graph*, as explained in more detail in the next section. A causal model is composed of the graph and a probability distribution that



Causality, Fig. 1 (**a**) A causal DAG. (**b**) Structure of the causal graph under some intervention that sets the value of A to a constant. (**c**) Structure of the causal graph under some intervention that changes the distribution of A

factorizes according to the graph, as in a standard graphical model. The only difference between a standard graphical model and a causal graphical model is that in the latter, extra assumptions are made. The graphical model can be seen as a way of encoding such assumptions.

The combination of assumptions and observational and interventional data generates such a graphical causal model. In the related problem of reinforcement learning, the agent has to maximize a specific utility function and typically has full control on which interventions (actions) can be performed. Here we will focus on the unsupervised problem of learning a causal model for a fixed input of observational and interventional data.

Because only some (or no) interventional data might be available, the learning system might not be able to answer some causal queries. That is, the system will not provide an answer for some prediction tasks.

Languages and Assumptions for Causal Inference

Directed acyclic graphs (DAGs) are a popular language in machine learning to encode qualitative statements about causal relationships. A DAG is composed of a set of vertices and a set of directed edges. The notions of parents, children, ancestors, and descendants are the usual ones found in graphical modeling literature.

In terms of causal statements, a directed edge $A \rightarrow B$ states that A is a *direct* cause of B: that is, different interventions on A will result on different distributions for B, even if we intervene on all other variables. The assumption that A is a cause of B is not used in non-causal graphical models.

A causal DAG G satisfies the *causal Markov condition* if and only if a vertex is independent of all of its non-descendants given its direct causes (parents). In Fig. 1a, A is independent of D, E, and F given its parents, B and C. It may or may not be independent of G given B and C.

The causal Markov condition implies several other conditional independence statements. For instance, in Fig. 1a, we have that H is independent of F given A. Yet, this is not a statement about the parents of any vertex. Pearl's d-separation criterion (Pearl 2000) is a sound and complete way of reading off independencies, out of a DAG, which are entailed by the causal Markov condition. We assume that the joint probability distribution over the vertex variables is *Markov* with respect to the graph, that is, any independence statement that is encoded by the graph should imply the corresponding independence in the distribution.

Representing Interventions

The local modularity given by the causal Markov condition leads to a natural notion of intervention. Random variable V, represented by a particular vertex in the graph, is following a *local mechanism*: its direct causes determine the distribution of V before its direct effects are generated. The role of an intervention is to *override* the natural local mechanism. An external agent substitutes the natural $P(V|Parents(V))$ by a new distribution $P_{Man}(V|Parents(V))$ while keeping the rest of the model unchanged ("Man" here stands for a particular manipulation). The notion of intervening by changing a single local mechanism is sometimes known as an *ideal intervention*. Other general types of interventions can be defined (Eaton and Murphy 2007), but the most common frameworks for calculating causal effects rely on this notion.

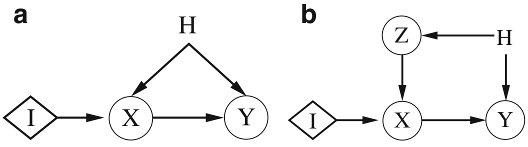
A common type of intervention is the point mass intervention, which happens when V is set to some constant v. This can be represented graphically by "wiping out" all edges into V. Figure 1b represents the resulting graph in (a) under a point manipulation of A. Notice that A is now d-separated from its direct causes under this regime. It is also probabilistically independent, since A is now constant. This allows for a graphical machinery that can read off independencies out of a *manipulated* graph (i.e., the one with removed edges). It is the idea of representing the natural state of the world with a single causal graph, and allowing for modifications in this graph according to the intervention of choice, that links the different regimes obtained under different interventions.

For the general case where a particular variable V is set to a new distribution, a *manipulation node* is added as an extra parent of V: this represents that an external agent is acting over that particular variable (Spirtes et al. 2000; Pearl 2000; Dawid 2003), as illustrated in Fig. 1c. $P(V|Parents(V))$ under intervention I is some new distribution $P_{Man}(V|Parents(V), I)$.

Calculating Distributions Under Interventions

The notion of independence is a key aspect of probabilistic graphical models, where it allows for efficient computation of marginal probabilities. In causal graphical models, it also fulfills another important role: independencies indicate that the effect of some interventions can be estimated using observational data.

We will illustrate this concept with a simple example. One of the key difficulties in calculating a causal effect is *unmeasured confounding*, i.e., hidden common causes. Consider Fig. 2a, where X is a direct cause of Y and H is a hidden common cause of both. I is an intervention vertex. Without extra assumptions, there is no way of estimating the effect of X on Y using a training set that is sampled from the observed marginal $P(X, Y)$. This is more easily seen in the case where the model is multivariate Gaussian with zero mean. In this case, each variable is a linear combination of its parents with standard Gaussian additive noise



Causality, Fig. 2 (**a**) X and Y have a hidden common cause H. (**b**) Y is dependent on the intervention node I given X, but conditioning on Z and marginalizing it out will allow us to eliminate the "backdoor" path that links X and Y through the hidden common cause H

$$\begin{aligned} X &= aH + \epsilon_X \\ Y &= bX + cH + \epsilon_Y \end{aligned}$$

where H is also a standard normal random variable. The manipulated distribution $P_{Man}(Y|X, I)$, where I is a point intervention setting $X = x$, is a Gaussian distribution with mean $b \cdot x$. Value x is given by construction, but one needs to learn the unknown value b.

One can verify that the covariance of X and Y in the natural state is given by $a + bc$. Observational data, i.e., data sampled from $P(X, Y)$, can be used to estimate the covariance of X and Y in the natural state, but from that it is not possible to infer the value of b: there are too many degrees of freedom.

However, there *are* several cases where the probability of $\mathbf{Y}$ given some intervention on $\mathbf{X}$ can be estimated with observational data and a given causal graph. Consider the graph in Fig. 2b. The problem again is to learn the distribution of Y given X under regime I, i.e., $P(Y|X, I)$. It can be read off from the graph that I and Y are not independent given X, which means $P(Y|X, I) \neq P(Y|X)$. How can someone then estimate $P(Y|X, I)$ if no data for this process has been collected? The answer lies on *reducing the "causal query" to a "probabilistic query"* where the dependence on I disappears (and, hence, the necessity of having data measured under the I intervention). This is done by relying on the assumptions encoded by the graph:

$$\begin{aligned} P(Y|X,I) &= \textstyle\sum_z P(Y|X,I,z)P(Z=z|X,I) && (Z \text{ is discrete in this example}) \\ &= \textstyle\sum_z P(Y|X,z)P(Z=z|X,I) && (Y \text{ and } I \text{ are independent given } Z) \\ &\propto \textstyle\sum_z P(Y|X,z)P(X|z,I)P(Z=z|I) && (\text{By Bayes' rule}) \\ &= \textstyle\sum_z P(Y|X,z)P(X|z,I)P(Z=z) && (Z \text{ and } I \text{ are marginally independent}) \end{aligned} \tag{1}$$

In the last line, we have $P(Y|X,Z)$ and $P(Z)$, which can be estimated with observational data, since no intervention variable I appears in the expression. $P(X|Z,I)$ is set by the external agent: its value is known by construction. This means that the causal distribution $P(Y|X,I)$ can be learned even in this case where X and Y share a hidden common cause H.

There are several notations for denoting an interventional distribution such as $P(Y|X,I)$. One of the earliest was due to Spirtes et al. (2000), which used the notation $P(Y|set\ X = x)$ to represent the distribution under an intervention I that fixed the value of X to some constant x. Pearl (2000) defines the operator *do* with an analogous purpose:

$$P(Y|do(X=x)) \tag{2}$$

Pearl's *do*-calculus is essentially a set of operations for reducing a probability distribution that is a function of some intervention to a probability distribution that does not refer to any intervention. All reductions are conditioned on the independencies encoded in a given causal graph. This is in the same spirit of the example presented above.

The advantage of such notations is that, for point interventions, they lead to simple yet effective transformations (or to a report that no transformation is possible). Spirtes et al. (2000) and Pearl (2000) provide a detailed account of such prediction tools. By making a clear distinction between $P(Y|X)$ (X under the natural state) and $P(Y|do(X))$ (X under some intervention), much of the confusion that conflates causal and noncausal predictions disappears.

It is important to stress that methods such as the *do*-calculus are nonparametric, in the sense that they rely on conditional independence constraints only. More informative reductions are possible if one is willing to provide extra information, such as assuming linearity of causal effects. For such cases, other parametric constraints can be exploited (Spirtes et al. 2000; Pearl 2000).

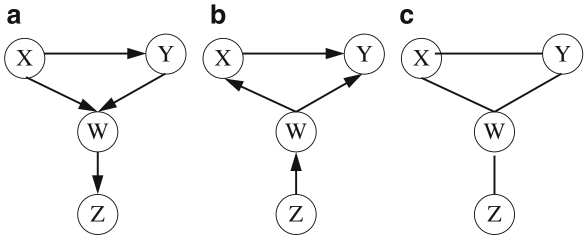
Learning Causal Structure

In all of the previous sections, we assumed that a causal graph was available. Since background knowledge is often limited, learning such graph structures becomes an important task. However, this cannot be accomplished without extra assumptions. To see why, consider again the example in Fig. 2a: if $a + bc = 0$, it follows that the X and Y are independent in the natural state. However, Y is *not* causally independent of X (if $b \neq 0$): $P(Y|do(X{=}x_1))$ and $P(Y|do(X{=}x_2))$ will be two different Gaussians with means $b \cdot x_1$ and $b \cdot x_2$, respectively.

This example demonstrates that an independence constraint that is testable by observational data does not warrant causal independence, at least based on the causal Markov condition only. However, an independence constraint that arises from particular identities such as $a + bc = 0$ is not *stable*, in the sense that it does not follow from the qualitative causal relations entailed by the Markov condition: a change in any of the parameter values will destroy such a constraint.

The artificiality of unstable independencies motivates an extra assumption: the *faithfulness* condition (Spirtes et al. 2000), also known as the *stability* condition (Pearl 2000). We say that a distribution P is faithful to a causal graph G if P is Markov with respect to G *and* if each conditional independence in P corresponds to some d-separation in G. That is, on top of the causal Markov condition, we assume that all independencies in P are entailed by the causal graph G.

The faithfulness condition allows us to reconstruct classes of causal graphs from observational data. In the simplest case, observing that X and Y are independent entails that there is no causal connection between X and Y. Conse-



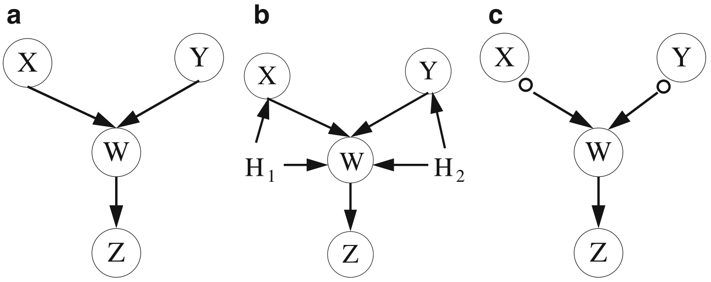
Causality, Fig. 3 (**a**) A particular causal graph which entails a few independence constraints, such as X and Z being independent given W. (**b**) A different causal graph that entails exactly the same independence constraints as in (**a**). (**c**) A representation for all graphs that entail the same conditional independencies as (**a**) and (**b**)

quently, $P(Y|do(X)) = P(Y|X) = P(Y)$. No interventional data was necessary to arrive at this conclusion, given the faithfulness condition.

In general, the solution is undetermined: more than one causal graph will be compatible with a set of observable independence constraints. Consider a simple example, where data is generated by a causal model with a causal graph given as in Fig. 3a. This graph entails some independencies, for instance, that X and Z are independent given W or that X and Y are not independent given any subset of $\{W, Z\}$. However, several other graphs entail the same conditional independences. The graph in Fig. 3b is one example. The learning task is then discovering an *equivalence class* of graphs, not necessarily a particular graph. This is in contrast with the problem of learning the structure of non-causal graphical models: the fact that there are other structures compatible with the data is not important in this case, since we will not use such graphical models to predict the effect of some hypothetical intervention. An equivalence class might not be enough information to reduce a desired causal query to a probabilistic query, but it might require much less prior knowledge than specifying a full causal graph.

Assume for now that no hidden common causes exist in this domain. In particular, the graphical object in Fig. 3c is a representation of the equivalence class of graphs that are compatible with the independencies encoded in Fig. 3a (Pearl 2000; Spirtes et al. 2000). All members of the equivalence class will have the same *skeleton* of this representation, i.e., the same adjacencies. An undirected edge indicates that there are two members in the equivalence class where directionality of this particular edge goes in opposite directions. Some different directions are illustrated in Fig. 3b. One can verify from the properties of d-separation that, if an expert or an experiment indicates that $X - W$ should be directed as $X \rightarrow W$, then the edge $W - Z$ is *compelled* to be directed as $W \rightarrow Z$: the direction $W \leftarrow Z$ is incompatible with the simultaneous findings that X and Z are independent given W and that X causes W .

More can be discovered if more independence constraints exist. In Fig. 4a, X is not a cause of Y. If we assume no hidden common causes exist in this domain, then no other causal graph is compatible with the independence constraints of Fig. 4a: the equivalence class is this graph only. However, the assumption of no hidden common causes is strong and undesirable. For instance, the graph in Fig. 4b, where H_1 and H_2 are hidden, is in the same equivalence class of (a). Yet, the graph in (a) indicates that $P(W|do(X)) = P(W|X)$, which can be arbitrarily different from the real $P(W|do(X))$ if Fig. 4b is the real graph. Some equivalence class representations, such as the Partial Ancestral Graph representation (Spirtes et al. 2000), are robust to hidden common causes: in Fig. 4c, an edge that has a circle as endpoint indicates that is not known if there is a causal path into both, e.g., X and W (which would be the case for a hidden common cause of X and W). The arrow into W does indicate that W cannot be a cause of X. A fully directed edge such as $W \rightarrow Z$ indicates total information: W is a cause of Z,



Causality, Fig. 4 (**a**) A particular causal graph with no other member on its equivalence class (assuming there are no hidden common causes). (**b**) Graph under the presence of two hidden common causes H_1 and H_2. (**c**) A representation for all graphs that entail the same conditional independencies as (**a**), without assuming the nonexistence of hidden common causes

Z is not a cause of W, and W and Z have no hidden common causes.

Given equivalence class representations and background knowledge, different types of algorithms explore independence constraints to learn an equivalence class. It is typically assumed that the true graph is acyclic. The basic structure is to evaluate how well a set of conditional independence hypotheses is supported by the data. Depending on which constraints are judged to hold in the population, we keep, delete, or orient edges accordingly. Some algorithms, such as the PC algorithm (Spirtes et al. 2000), test a single independence hypothesis at a time and assemble the individual outcomes in the end into an equivalence class representation. Other algorithms such as the GES algorithm (Meek 1997; Chickering 2002) start from a prior distribution for graphs and parameters and proceed to compare the marginal likelihood of members of different equivalence classes (which can be seen as a Bayesian joint test of independence hypotheses). In the end, this reduces to a search for the maximum a posteriori equivalence class estimator. Both PC and GES have consistency properties: in the limit of infinite data, they return the right equivalence class under the faithfulness assumption. However, both PC and GES, and most causal discovery algorithms, assume that there are no hidden common causes in the domain. The Fast Causal Inference (FCI) algorithm of Spirtes et al. (2000) is able to generate equivalence class representations as in Fig. 4c. As in the PC algorithm, this is done by testing a single independence hypothesis at a time and therefore is a high-variance estimator given small samples. A GES-like algorithm with the consistency properties of FCI is not currently known. An algorithm that allows for cyclic networks is discussed by Richardson (1996).

Semiparametric Models

Our examples relied on conditional independence constraints. In this case, the equivalence class is known as the *Markov equivalence class*. Markov equivalence classes are “nonparametric,” in the sense that they do not refer to any particular probability family. In practice, this advantage is limited by our ability to test independence hypotheses within flexible probability families. Another shortcoming of Markov equivalence classes is that they might be poorly informative if few independence constraints exist in the population. This will happen, for instance, if a single hidden variable is a common cause of all observed variables. If one is willing to incorporate further assumptions, such as linearity of causal relationships, semiparametric constraints can be used to define other types of equivalence classes that are more discriminative than the Markov equivalence class. Silva et al. (2006) describe how some rank constraints in the covariance matrix of the observed variables can be used to learn the structure of linear models, even if no independence constraints are observable. Shimizu et al. (2006) provide a solution to find the causal ordering of a linear DAG model without latent variables, by exploiting information beyond the second moments of a distribution in the non-Gaussian case. Entner et al. (2012) introduce an approach to estimate causal effects in non-Gaussian linear systems under some assumptions of directionality but allowing for unmeasured confounding. Peters et al. (2014) develop a general method for learning directionality in nonlinear models with additive noise.

Confidence Intervals

Several causal learning algorithms such as the PC and FCI algorithms (Spirtes et al. 2000) are consistent, in the sense that they can recover the correct equivalence class given the faithfulness assumption and an infinite amount of data. Although point estimates of causal effects are important, it is also important to provide confidence intervals. From a frequentist perspective, it has been shown that this is not possible given the faithfulness assumption only (Robins et al. 2003). An intuitive explanation is as follows: consider the model such as the one in Fig. 2a. For any given sample size, there is at least one model such that the associations due to the paths $X \leftarrow H \rightarrow Y$ and $X \rightarrow Y$ nearly cancel each other (faithfulness is still preserved), making the covariance of X and Y statistically indistinguishable from zero. In order to achieve uniform consistency, causal inference algorithms need assumptions stronger than faithfulness. Zhang and Spirtes (2003) provide some directions.

Other Languages and Tasks in Causal Learning

A closely related language for representing causal models is the *potential outcomes* framework popularized by Donald Rubin (Rubin 2005). In this case, random variables for a same variable Y are defined for each possible state of the intervened variable X. Notice that, by definition, only one of the possible Y outcomes can be observed for any specific data point. This framework is popular in the statistics literature as a type of missing data model. The relation between potential outcomes and several other representations of causality is discussed by Richardson and Robins (2013).

A case where potential outcomes become particularly motivated is in *causal explanation*. In this setup, the model is asked for the probability that a particular event in time was the cause of a particular outcome. This is often cast as a *counterfactual question*: had A been false, would B still have happened? Questions in history and law are of this type: the legal responsibility of an airplane manufacturer in an accident depends on technical malfunction being an *actual cause* of the accident. Ultimately, such issues of causal explanation, actual causation and other counterfactual answers, are untestable. Although machine learning can be a useful tool to derive the consequences of assumptions combined with data about other events of the same type, in general the answers will not be robust to changes in the assumptions, and the proper assumptions ultimately cannot be selected with the available data. Some advances in generating explanations with causal models are described by Halpern and Pearl (2005).

Recommended Reading

Chickering D (2002) Optimal structure identification with greedy search. J Mach Learn Res 3:507–554

Cooper G, Yoo C (1999) Causal discovery from a mixture of experimental and observational data. In: Proceedings of the 15th conference on uncertainty in artificial intelligencem (UAI-1999), Stockholm, pp 116–125

Dawid AP (2003) Causal inference using influence diagrams: the problem of partial compliance. In: Green PJ, Hjort NL, Richardson S (eds) Highly structured stochastic systems. Oxford University Press, New York, pp 45–65

Eaton D, Murphy K (2007) Exact Bayesian structure learning from uncertain interventions. In: Proceedings of the 11th international conference on artificial intelligence and statistics (AISTATS-2007), San Juan, pp 107–114

Entner D, Hoyer PO, Spirtes P (2012) Statistical test for consistent estimation of causal effects in linear non-gaussian models. In: Proceedings of the 15th international conference on artificial intelligence and statistics (AISTATS-2012), La Palma, pp 364–372

Halpern J, Pearl J (2005) Causes and explanations: a structural-model approach. Part II: explanations. Br J Philos Sci 56:889–911

Hyttinen A, Eberhardt F, Hoyer PO (2013) Experiment selection for causal discovery. J Mach Learn Res 14:3041–3071

Meek C (1997) Graphical models: selecting causal and statistical models. PhD thesis, Carnegie Mellon University

Pearl J (2000) Causality: models, reasoning and inference. Cambridge University Press, New York

Peters J, Mooij JM, Janzing D, Schölkopf B (2014) Causal discovery with continuous additive noise models. J Mach Learn Res 15:2009–2053

Richardson TS (1996) A discovery algorithm for directed cyclic graphs. In: Proceedings of 12th conference on uncertainty in artificial intelligence, Portland

Richardson TS, Robins J (2013) Single world intervention graphs (SWIGs): a unification of the counterfactual and graphical approaches to causality. Working Paper Number 128, Center for Statistics and the Social Sciences, University of Washington

Robins J, Scheines R, Spirtes P, Wasserman L (2003) Uniform consistency in causal inference. Biometrika 90:491–515

Rosenbaum P (2002) Observational studies. Springer, New York

Rubin D (2005) Causal inference using potential outcomes: design, modeling, decisions. J Am Stat Assoc 100(469):322–331

Sachs K, Perez O, Pe'er D, Lauffenburger D, Nolan G (2005) Causal protein-signaling networks derived from multiparameter single-cell data. Science 308:523–529

Shimizu S, Hoyer P, Hyvärinen A, Kerminen A (2006) A linear non-Gaussian acyclic model for causal discovery. J Mach Learn Res 7:2003–2030

Silva R, Scheines R, Glymour C, Spirtes P (2006) Learning the structure of linear latent variable models. J Mach Lear Res 7:191–246

Spirtes P, Glymour C, Scheines R (2000) Causation, prediction and search. MIT Press, Cambridge

Wasserman L (2004) All of statistics. Springer, New York

Zhang J, Spirtes P (2003) Strong faithfulness and uniform consistency in causal inference. In: Proceedings of the 19th conference in uncertainty in artificial intelligence (UAI-2013), Acapulco, pp 632–639

CC

▶ Cascade Correlation

Certainty Equivalence Principle

▶ Internal Model Control

Characteristic

▶ Attribute

Citation or Reference Matching (When Applied to Bibliographic Data)

▶ Record Linkage

City Block Distance

▶ Manhattan Distance

Class

Chris Drummond
National Research Council of Canada, Ottawa, ON, Canada

Synonyms

Category; Collection; Kind; Set; Sort; Type

Definition

A class is a collection of things that might reasonably be grouped together. Classes that we commonly encounter have simple names so, as humans, we can easily refer to them. The class of dogs, for example, allows me to say "my dog ate my newspaper" without having to describe a particular dog, or indeed, a particular newspaper. In machine learning, the name of the class is called the class label. Exactly what it means to belong to a class, or category, is a complex philosophical question but often we think of a class in terms of the common properties of its members. We think particularly of those properties which seperate them from other things which are in many ways similar, e.g., cats mieow and dogs bow-wow. We would be unlikely to form a class from a random collection of things, as they would share no common properties. Knowing something belonged to such a collection would be of no particular benefit. Although many every day classes will have simple names, we may construct them however we like, e.g., "The things I like to eat for breakfast on a Saturday morning." As there is no simple name for such a collection, in machine learning we would typically refer to it as the positive class, and all occurences of it are positive examples; the negative class would be everything else.

Motivation and Background

The idea of a class is important in learning. If we discover something belongs to a class, we suddenly know quite a lot about it even if we have not encountered that particular example before. In machine learning, our use of the term accords closely with the mathematical definition of a class, as a collection of sets unambiguously defined by a property that all its members share. It also accords with the idea of equivalence classes, which group similar things. Sets have an intension, the description of what it means to be a member, and an extension, things that belong to the set, useful properties of a class in machine learning. Class is also a term used extensively in knowledge bases to denote an important relationship between groups, of sub-class and super class. Learning is often viewed as a way of solving the knowledge acquisition bottleneck (Buchanan et al. 1983) in knowledge bases and the use of the term class in machine learning highlights this connection.

Recommended Reading

Buchanan B, Barstow D, Bechtel R, Bennett J, Clancey W, Kulikowski C et al (1983) Constructing an expert system. In: Hayes-Roth F, Waterman DA, Lenat DB (eds) Building expert systems. Addison-Wesley, Reading, pp 127–167

Class Binarization

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt, Darmstadt, Deutschland
Department of Information Technology, University of Leoben, Leoben, Austria

Abstract

Many learning algorithms are only designed to separate two classes from each other. For example, ▶ concept-learning algorithms assume positive examples and negative examples (counterexamples) for the concept to learn, and many statistical learning techniques, such as neural networks or ▶ support vector machines, can only find a single separating decision surface. One way to apply these algorithms to multi-class problem is to transform the original multi-class problem into multiple binary problems.

Synonyms

Error-correcting output codes (ECOC); One-against-all training; One-against-one training; Pairwise classification

Methods

The best-known techniques are:

One against all: one concept-learning problem is defined for each class, i.e., each class is in turn used as the positive class, and all other classes form the negative class.

Pairwise (One against one): one concept is learned for each pair of classes (Fürnkranz 2002). This may be viewed as a special case of ▶ preference learning.

Error-correcting output codes: ECOC allow arbitrary subsets of the classes to form the positive and negative classes of the binary problems. In the original formulation (Dietterich and Bakiri 1995), all classes have to be used for each problem, a later generalization (Allwein et al. 2000) allows arbitrary combinations. Clearly, one against all and one against one are special cases of ECOC.

The predictions of the binary classifiers must then be combined into an overall prediction. Commonly used techniques include voting and finding the nearest neighbor in the ECOC decoding matrix (Allwein et al. 2000).

Cross-References

▶ Preference Learning
▶ Rule Learning

Recommended Reading

Allwein EL, Schapire RE, Singer Y (2000) Reducing multiclass to binary: a unifying approach for margin classifiers. J Mach Learn Res 1: 113–141

Dietterich TG, Bakiri G (1995) Solving multiclass learning problems via error-correcting output codes. J Artif Intell Res 2:263–286

Fürnkranz J (2002) Round robin classification. J Mach Learn Res 2:721–747. http://www.ai.mit.edu/projects/jmlr/papers/volume2/fuernkranz02a/html/.

Class Imbalance Problem

Charles X. Ling and Victor S. Sheng
The University of Western Ontario, London, ON, Canada

Definition

Data are said to suffer the *Class Imbalance Problem* when the class distributions are highly imbalanced. In this context, many ▶ classification learning algorithms have low predictive accuracy for the infrequent class. ▶ Cost-sensitive learning is a common approach to solve this problem.

Motivation and Background

Class imbalanced datasets occur in many real-world applications where the class distributions of data are highly imbalanced. For the two-class case, without loss of generality, one assumes that the minority or rare class is the positive class, and the majority class is the negative class. Often the minority class is very infrequent, such as 1 % of the dataset. If one applies most traditional (cost-insensitive) classifiers on the dataset, they are likely to predict everything as negative (the majority class). This was often regarded as a problem in learning from highly imbalanced datasets.

However, Provost (2000) describes two fundamental assumptions that are often made by traditional cost-insensitive classifiers. The first is that the goal of the classifiers is to maximize the accuracy (or minimize the error rate); the second is that the class distribution of the training and test datasets is the same. Under these two assumptions, predicting everything as negative for a highly imbalanced dataset *is often the right thing to do*. Drummond and Holte (2005) show that it is usually very difficult to outperform this simple classifier in this situation.

Thus, the imbalanced class problem becomes meaningful only if one or both of the two assumptions above are not true; that is, if the cost of different types of error (false positive and false negative in the binary classification) is not the same, or if the class distribution in the test data is different from that of the training data. The first case can be dealt with effectively using methods in cost-sensitive meta-learning (see ▶ Cost-sensitive learning).

In the case when the misclassification cost is not equal, it is usually more expensive to misclassify a minority (positive) example into the majority (negative) class, than a majority example into the minority class (otherwise it is more plausible to predict everything as negative). That is, *FNcost* > *FPcost*. Thus, given the values of *FNcost* and *FPcost*, a variety of cost-sensitive meta-learning methods can be, and have been, used to solve the class imbalance problem (Japkowicz and Stephen 2002; Ling and Li 1998). If the values of *FNcost* and *FPcost* are not unknown explicitly, *FNcost* and *FPcost* can be assigned to be proportional to the number of positive and negative training cases (Japkowicz and Stephen 2002).

In case the class distributions of training and test datasets are different (e.g., if the training data is highly imbalanced but the test data is more balanced), an obvious approach is to sample the training data such that its class distribution is the same as the test data. This can be achieved by oversampling (creating multiple copies of examples of) the minority class and/or undersampling (selecting a subset of) the majority class (Provost 2000).

Note that sometimes the number of examples of the minority class is too small for classifiers to learn adequately. This is the problem of insufficient (small) training data and different from that of imbalanced datasets.

Recommended Reading

Drummond C, Holte R (2000) Exploiting the cost (in)sensitivity of decision tree splitting criteria. In: Proceedings of the seventeenth international conference on machine learning, Stanford, pp 239–246

Drummond C, Holte R (2005) Severe class imbalance: why better algorithms aren't the answer. In: Proceedings of the sixteenth European conference of machine learning, Porto, vol 3720. LNAI, pp 539–546

Japkowicz N, Stephen S (2002) The class imbalance problem: a systematic study. Intell Data Anal 6(5):429–450

Ling CX, Li C (1998) Data mining for direct marketing – specific problems and solutions. In: Proceedings of fourth international conference on knowledge discovery and data mining (KDD-98), New York City, pp 73–79

Provost F (2000) Machine learning from imbalanced data sets 101. In: Proceedings of the AAAI'2000 workshop on imbalanced data

Classification

Chris Drummond
National Research Council of Canada, Ottawa, ON, Canada

Synonyms

Categorization; Generalization; Identification; Induction; Recognition

Definition

In common usage, the word classification means to put things into categories, group them together in some useful way. If we are screening for a disease, we would group people into those with the disease and those without. We, as humans, usually do this because things in a group, called a ▶ class in machine learning, share common characteristics. If we know the class of something, we know a lot about it. In machine learning, the term classification is most commonly associated with a particular type of learning where examples of one or more ▶ classes, labeled with the name of the class, are given to the learning algorithm. The algorithm produces a classifier which maps the properties of these examples, normally expressed as ▶ attribute-value pairs, to the class labels. A new example whose class is unknown is classified when it is given a class label by the classifier based on its properties. In machine learning, we use the word classification because we call the grouping of things a class. We should note, however, that other fields use different terms. In philosophy and statistics, the term categorization is more commonly used. In many areas, in fact, classification often refers to what is called clustering in machines learning.

Motivation and Background

Classification is a common, and important, human activity. Knowing something's class allows us to predict many of its properties and so act appropriately. Telling other people its class allows them to do the same, making for efficient communication. This emphasizes two commonly held views of the objectives of learning. First, it is a means of ▶ generalization, to predict accurately the values for previously unseen examples. Second, it is a means of compression, to make transmission or communication more efficient. Classification is certainly not a new idea and has been studied for some considerable time. From the days of the early Greek philosophers such as Socrates, we had the idea of categorization. There are essential properties of things that make them what they are. It embodies the idea that there are natural kinds, ways of grouping things, that are inherent in the world. A major goal of learning, therefore, is recognizing natural kinds, establishing the necessary and sufficient conditions for belonging to a category. This "classical" view of categorization, most often attributed to Aristotle, is now strongly disputed. The main competitor is prototype theory; things are categorized by their similarity to a prototypical example (Lakoff 1987), either real or imagined. There is also much debate in psychology (Ashby and Maddox 2005), where many argue that there is no single method of categorization used by humans.

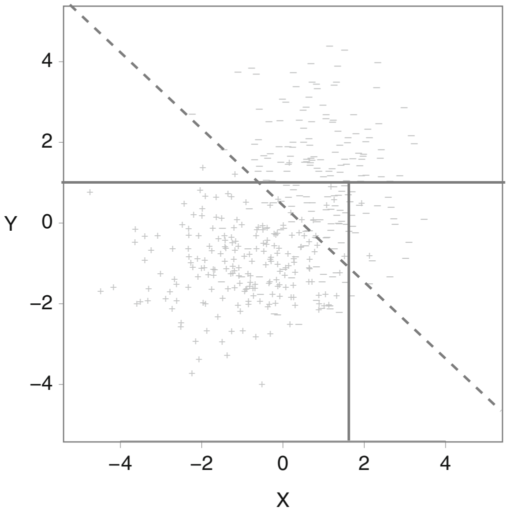
As much of the inspiration for machine learning originated in how humans learn, it is unsurprising that our algorithms reflect these distinctions. ▶ Nearest neighbor algorithms would seem to have much in common with prototype theory. These have been part of pattern recognition for some time (Cover and Hart 1967) and have become popular in machine learning, more recently, as ▶ instance-based learners (Aha et al. 1991). In machine learning, we measure the distance to one or more members of a concept rather a specially constructed prototype. So, this type of learning is perhaps more a case of the exemplar learning found in the psychological literature, where multiple examples represent a category. The closest we have to prototype learning occurs in clustering, a type of ▶ unsupervised learning, rather than classification. For example, in ▶ k-means clustering group membership is determined by closeness to a central value.

In the early days of machine learning, our algorithms (Mitchell 1977; Winston 1975) had much in common with the classical theory of categorization in philosophy and psychology. It was assumed that the data were consistent, there were no examples with the same attribute values but belonging to different classes. It was quickly realized that, even if the properties where necessary and sufficient to capture the class, there was often noise in the attribute and perhaps the class values. So, complete consistency was seldom attainable in practice. New ▶ classification algorithms were designed, which could tolerate some noise, such as ▶ decision trees (Breiman et al. 1984; Quinlan 1986, 1993) and rule-based learners (see ▶ Rule Learning) (Clark and Niblett 1989; Holte 1993; Michalski 1983).

Structure of the Learning System

Whether one uses instance-based learning, rule-based learning, decision trees, or indeed any other classification algorithm, the end result is the division of the input space into regions belonging to a single class. The input space is defined by the Cartesian product of the attributes, all possible combinations of possible values.

As a simple example, Fig. 1 shows two classes + and −, each a random sample of a normal distribution. The attributes are X and Y of real type. The values for each attribute range from $\pm\infty$. The figure shows a couple of alternative ways that the space may be divided into regions. The bold dark lines, construct regions using lines that are parallel to the axes. New examples that have Y less than 1 and X less than 1.5 with be classified as +, all others classified as −. Decision trees and rules form this type of boundary. A ▶ linear discriminant function, such as the bold dashed line, would divide the space into half-spaces, with new examples below the line being classified as + and those above as −. Instance-based learning will also divide the space into re-



Classification, Fig. 1 Dividing the input space gions but the boundary is implicit. Classification occurs by choosing the class of the majority of the nearest neighbors to a new example. To make the boundary explicit, we could mark the regions where an example would be classified as + and those classified as −. We would end up with regions bounded by polygons.

What differs among the algorithms is the shape of the regions, and how and when they are chosen. Sometimes the regions are implicit as in lazy learners (see ▸ Lazy Learning) (Aha 1997), where the boundaries are not decided until a new example is being classified. Sometimes the regions are determined by decision theory as in generative classifiers (see ▸ Generative Learning) (Rubinstein and Hastie 1997), which model the full joint distribution of the classes. For all classifiers though, the input space is effectively partitioned into regions representing a single class.

Applications

One of the reasons that classification is an important part of machine learning is that it has proved to be a very useful technique for solving practical problems. Classification has been used to help scientists in the exploration, and comprehension, of their particular domains of interest. It has also been used to help solve significant industrial problems. Over the years a number of authors have stressed the importance of applications to machine learning and listed many successful examples (Brachman et al. 1996; Langley and Simon 1995; Michie 1982). There have also been workshops on applications (Kodratoff 1994; Aha and Riddle 1995; Engels et al. 1997) at major machine learning conferences and a special issue of Machine Learning (Kohavi and Provost 1998), one of the main journals in the field. There are now conferences that are highly focused on applications. Collocated with major artificial intelligence conferences is the Innovative Applications of Artificial Intelligence conference. Since 1989, this conference has highlighted practical applications of machine learning, including classification (Schorr and Rappaport 1989). In addition, there are now at least two major knowledge discovery and data mining conferences (Fayyad and Uthurusamy 1995; Komorowski and Zytkow 1997) with a strong focus on applications.

Future Directions

In machine learning, there are already a large number of different classification algorithms, yet new ones still appear. It seems unlikely that there is an end in sight. The “no free lunch theory” (Wolpert and Macready 1997) indicates that there will never be a single best algorithm, better than all others in terms of predictive power. However, apart from their predictive performance, each classifier has its own attractive properties which are important to different groups of people. So, new algorithms are still of value. Further, even if we are solely concerned about performance, it may be useful to have many different algorithms, all with their own biases (see ▸ Inductive Bias). They may be combined together to form an ensemble classifier (Caruana et al. 2004), which outperforms single classifiers of one type (see ▸ Ensemble Learning).

Limitations

Classification has been critical part of machine research for some time. There is a concern that the emphasis on classification, and more generally on ▸ supervised learning, is too strong. Certainly much of human learning does not use, or require, labels supplied by an expert. Arguably, unsupervised learning should play a more central role in machine learning research. Although classification does require a label, it does necessarily need an expert to provide labeled examples. Many successful applications rely on finding some, easily identifiable, property which stands in for the class.

Recommended Reading

Aha DW (1997) Editorial. Artif Intell Rev 11(1–5):1–6

Aha DW, Riddle PJ (eds)(1995) Workshop on applying machine learning in practice. In: Proceedings of the

12th international conference on machine learning, Tahoe City

Aha DW, Kibler D, Albert MK (1991) Instance-based learning algorithms. Mach Learn 6(1):37–66

Ashby FG, Maddox WT (2005) Human category learning. Ann Rev Psychol 56:149–178

Bishop CM (2007) Pattern recognition and machine learning. Springer, New York

Brachman RJ, Khabaza T, Kloesgen W, Piatetsky-Shapiro G, Simoudis E (1996) Mining business databases. Commun ACM 39(11):42–48

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. Wadsworth, Belmont

Caruana R, Niculescu-Mizil A, Crew G, Ksikes A (2004) Ensemble selection from libraries of models. In: Proceedings of the 21st international conference on machine learning, Banff, pp 137–144

Clark P, Niblett T (1989) The CN2 induction algorithm. Mach Learn 3:261–284

Cover T, Hart P (1967) Nearest neighbor pattern classification. IEEE Trans Inf Theory 13:21–27

Dietterich T, Shavlik J (eds) Readings in machine learning. Morgan Kaufmann, San Mateo

Engels R, Evans B, Herrmann J, Verdenius F (eds) (1997) Workshop on machine learning applications in the real world; methodological aspects and implications. In: Proceedings of the 14th international conference on machine learning, Nashville

Fayyad UM, Uthurusamy R (eds)(1995) Proceedings of the first international conference on knowledge discovery and data mining, Montreal

Holte RC (1993) Very simple classification rules perform well on most commonly used datasets. Mach Learn 11(1):63–91

Kodratoff Y (ed)(1994) Proceedings of MLNet workshop on industrial application of machine learning, Douran

Kodratoff Y, Michalski RS (1990) Machine learning: an artificial intelligence approach, vol 3. Morgan Kaufmann, San Mateo

Kohavi R, Provost F (1998) Glossary of terms. Editorial for the special issue on applications of machine learning and the knowledge discovery process. Mach Learn 30(2/3)

Komorowski HJ, Zytkow JM (eds) (1997) Proceedings of the first European conference on principles of data mining and knowledge discovery

Lakoff G (1987) Women, fire and dangerous things. University of Chicago Press, Chicago

Langley P, Simon HA (1995) Applications of machine learning and rule induction. Commun ACM 38(11):54–64

Michalski RS (1983) A theory and methodology of inductive learning. In: Michalski RS, Carbonell TJ, Mitchell TM (eds) Machine learning: an artificial intelligence approach. TIOGA Publishing, Palo Alto, pp 83–134

Michalski RS, Carbonell JG, Mitchell TM (eds) (1983) Machine learning: an artificial intelligence approach. Tioga Publishing Company, Palo Alto

Michalski RS, Carbonell JG, Mitchell TM (eds) (1986) Machine learning: an artificial intelligence approach, vol 2. Morgan Kaufmann, San Mateo

Michie D (1982) Machine intelligence and related topics. Gordon and Breach Science Publishers, New York

Mitchell TM (1977) Version spaces: a candidate elimination approach to rule learning. In: Proceedings of the fifth international joint conferences on artificial intelligence, Cambridge, pp 305–310

Mitchell TM (1997) Machine learning. McGraw-Hill, Boston

Quinlan JR (1986) Induction of decision trees. Mach Learn 1:81–106

Quinlan JR (1993) C4.5 programs for machine learning. Morgan Kaufmann, San Mateo

Rubinstein YD, Hastie T (1997) Discriminative vs informative learning. In: Proceedings of the third international conference on knowledge discovery and data mining, Newport Beach, pp 49–53

Russell S, Norvig P (2003) Artificial intelligence: a modern approach. Prentice-Hall, Upper Saddle River

Schorr H, Rappaport A (eds) (1989) Proceedings of the first conference on innovative applications of artificial intelligence, Stanford

Winston PH (1975) Learning structural descriptions from examples. In: Winston PH (ed) The psychology of computer vision. McGraw-Hill, New York, pp 157–209

Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques. Morgan Kaufmann, San Fransisco

Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82

Classification Algorithms

There is a very large number of classification algorithms, including ▸ decision trees, ▸ instance-based learners, ▸ support vector machines, ▸ rule-based learners, ▸ neural networks, ▸ Bayesian networks. There also ways of combining them into ensemble classifiers such as ▸ boosting, ▸ bagging, ▸ stacking, and forests of trees.

To delve deeper into classifiers and their role in machine learning, a number of books are recommended covering machine learning (Bishop 2007; Mitchell 1997; Witten and Frank 2005) and artificial intelligence (Russell and Norvig 2003) in general. Seminal papers on classifiers can be found in collections of papers on machine learning (Dietterich and Shavlik 1990; Kodratoff and Michalski 1990; Michalski et al. 1983, 1986).

Recommended Reading

Bishop CM (2007) Pattern recognition and machine learning. Springer, New York

Dietterich T, Shavlik J (eds) (1990) Readings in machine learning. Morgan Kaufmann, San Mateo

Kodratoff Y, Michalski RS (1990) Machine learning: an artificial intelligence approach, vol 3. Morgan Kaufmann, San Mateo

Michalski RS, Carbonell JG, Mitchell TM (eds) (1983) Machine learning: an artificial intelligence approach. Tioga Publishing Company, Palo Alto

Michalski RS, Carbonell JG, Mitchell TM (eds) (1986) Machine learning: an artificial intelligence approach, vol 2. Morgan Kaufmann, San Mateo

Mitchell TM (1997) Machine learning. McGraw-Hill, Boston

Russell S, Norvig P (2003) Artificial intelligence: a modern approach. Prentice-Hall, Upper Saddle River

Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques. Morgan Kaufmann, San Fransisco

Classification Learning

▶ Concept Learning

Classification Rule

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt,
Darmstadt, Deutschland
Department of Information Technology,
University of Leoben, Leoben, Austria

Abstract

A classification rule is an `IF-THEN` rule. The condition of the rule (the *rule body* or *antecedent*) typically consists of a conjunction of Boolean terms, each one constituting a constraint that needs to be satisfied by an example. If all constraints are satisfied, the rule is said to *fire*, and the example is said to be *covered* by the rule. The *rule head* (also called the *consequent* or *conclusion*) consists of a single ▶ class value, which is predicted in case the rule fires. This is in contrast to ▶ association rules, which allow multiple features in the head.

Method

Typical terms consist of tests for the presence of a particular ▶ attribute value or, in the case of numerical attributes, of an inequality that requires that the observed value is above or below a threshold. More expressive constraints include *set-valued attributes* (several values of the same attribute can be observed in the training examples), *internal disjunctions* (only one of several values of the same attribute needs to present), *hierarchical attributes* (certain values of the attributes subsume other values), etc.

Conjunctive combinations of features may be viewed as statements in ▶ propositional logic (▶ propositional rules). If relations between features can be considered (i.e., if propositions can be formulated in ▶ first-order logic), we speak of ▶ first-order rules.

Cross-References

▶ Association Rule
▶ Decision List
▶ First-Order Logic
▶ Propositional Logic
▶ Rule Learning

Classification Tree

▶ Decision Tree

Classifier Calibration

Peter A. Flach
Department of Computer Science, University of Bristol, Bristol, UK

Abstract

Classifier calibration is concerned with the scale on which a classifier's scores are expressed. While a classifier ultimately maps instances to discrete classes, it is often beneficial to decompose this mapping into a *scoring classifier* which outputs one or more real-valued numbers and a *decision rule* which converts these numbers into predicted classes. For example, a linear classifier might output a positive or negative score whose magnitude is proportional to the distance between the instance and the *decision boundary*, in which case the decision rule would be a simple threshold on that score. The advantage of calibrating these scores to a known, domain-independent scale is that the decision rule then also takes a domain-independent form and does not have to be learned. The best-known example of this occurs when the classifier's scores approximate, in a precise sense, the ▸ posterior probability over the classes; the main advantage of this is that the optimal decision rule is to predict the class that minimizes expected cost averaged over all possible true classes. The main methods to obtain calibrated scores are *logistic calibration*, which is a parametric method that assumes that the distances on either side of the decision boundary are normally distributed and a nonparametric alternative that is variously known as *isotonic regression*, the *pool adjacent violators* (PAV) method or the *ROC convex hull* (ROCCH) method.

Synonyms

Isotonic calibration; Logistic calibration; Probability calibration; Sigmoid calibration

Motivation and Background

A predictive model can be said to be well calibrated if its predictions match observed distributions in the data. In particular, a probabilistic classifier is well calibrated if, among the instances receiving a predicted probability vector p, the class distribution is approximately distributed as p. Hence, the classifier approximates, in some sense, the class posterior, although the approximation can be crude: for example, a constant classifier predicting the overall class distribution for every instance is perfectly calibrated in this sense.

Calibration is closely related to optimal decision making and cost-sensitive classification, where we wish to determine the predicted class that minimizes expected misclassification cost averaged over all possible true classes. The better our estimates of the class posterior are, the closer we get to the (irreducible) Bayes risk. A sufficiently calibrated classifier can be simply thresholded at a threshold directly derived from the misclassification costs. Similar thresholds can be used to optimally adapt to a change in class prior or to a combination of both.

Some training algorithms naturally yield well-calibrated classifiers, including ▸ logistic regression and ▸ decision trees (with Laplace smoothing but without pruning; Provost and Domingos 2003; Ferri et al. 2003). Others do not take sufficient account of distributional factors (e.g., ▸ support vector machines) or make unrealistic assumptions (e.g., ▸ naive Bayes) and need to be calibrated in post-processing. Well-established calibration methods include logistic calibration (parametric) and the ROC convex hull method, also known as pair-adjacent violators and isotonic regression (nonparametric).

In order to evaluate the quality of probability estimates, we can use the Brier score, which measures mean squared deviation from the "ideal" probabilities 1 and 0. The Brier score can be decomposed into refinement loss and calibration loss, where the former assesses the likelihood of instances from different classes receiving the same probability estimate, and the latter measures the mean squared deviation from the empirical probabilities. Both quantities can be further decomposed: e.g., the refinement loss has model-dependent and model-independent components (grouping loss and irreducible loss), and the calibration loss has a component that can be reduced to zero by a simple affine transformation of the scores (Kull and Flach 2015). These losses and decompositions can be visualized by means of various cost curves.

One topic of current interest is that the above view of classifier calibration is closely tied to a particular family of losses (error rate and cost-sensitive variants). Class posteriors are the right thing to model for this family, but not for others. For example, it is known that thresholds on the class posterior that are optimal for F1-score are lower than optimal thresholds for accuracy, but there is no one-to-one mapping between the two. The cost-sensitive perspective offers an alternative view of calibration: a classifier is well calibrated if it outputs, for every instance, the cost parameter for which that instance is on (or close to) the decision boundary. This naturally leads to the idea of classifiers outputting several scores, each calibrated for a different cost model (e.g., accuracy and F1-score).

Solutions

We start by demonstrating that better approximation of posterior probabilities allows the classifier to make better predictions in a decision-theoretic sense (see also ▸ Cost-Sensitive Classification). Denote the cost of predicting class j for an instance of true class i as $C(\hat{Y} = j|Y = i)$. Since we don't know the true class of an unlabelled instance, we need to base our prediction on an assessment of the expected cost over all possible true classes. The (true) expected cost of predicting class j for instance x is

$$\begin{aligned} C(\hat{Y} = j|X = x) \\ = \sum_i P(Y = i|X = x)C(\hat{Y} = j|Y = i) \end{aligned} \tag{1}$$

where $P(Y = i|X = x)$ is the probability of instance x having true class i. The optimal decision is then to predict the class which minimizes expected cost:

$$\begin{aligned} \hat{Y}^* &= \arg\min_j C(\hat{Y} = j|X = x) \\ &= \arg\min_j \sum_i P(Y{=}i|X{=}x)C(\hat{Y}{=}j|Y{=}i) \end{aligned}$$

In the special case that all misclassifications have equal cost, we can assume without loss of generality that $C(\hat{Y} = j|Y = i) = 1$ for $i \neq j$ and $C(\hat{Y} = j|Y = i) = 0$ for $i = j$, which gives

$$\begin{aligned} C(\hat{Y} = j|X = x) &= \sum_{i \neq j} P(Y = i|X = x) \\ &= 1 - P(Y = j|X = x) \\ \hat{Y}^* &= \arg\min_j [1 - P(Y = j|X = x)] \\ &= \arg\max_j P(Y = j|X = x) \end{aligned}$$

The main point is that *knowing the true class posterior allows the classifier to make optimal decisions*, either in a cost-sensitive or cost-indifferent setting. It therefore makes sense for a classifier to (approximately) learn the true class posterior. (Notice that this crucially depends on the cost model expressed by Eq. (1), which assumes that our aim is to optimize a cost-based version of accuracy. A different cost model will require a different notion of calibrated score, as will be briefly considered later for the case of the F_β score.) For the remainder of this entry, we will concentrate on binary classification, returning to the challenges of multiclass calibration at the end.

Optimal Decision Thresholds

In binary classification we have the following expected costs for positive and negative predictions:

$$\begin{aligned} C(\hat{Y} = +|X = x) &= P(+|x)C(+|+) \\ &\quad + (1 - P(+|x))C(+|-) \\ C(\hat{Y} = -|X = x) &= P(+|x)C(-|+) \\ &\quad + (1 - P(+|x))C(-|-) \end{aligned}$$

where $P(+|x)$ is shorthand for $P(Y{=}+|X{=}x)$ and $C(j|i)$ for $C(\hat{Y} = j|Y = i)$. On the optimal decision boundary, these two expected costs are equal, which gives

$$P(+|x)C(+|+) + (1 - P(+|x))C(+|-) \\ = P(+|x)C(-|+) + (1 - P(+|x))C(-|-)$$

and so

$$P(+|x) \\ = \frac{C(+|-) - C(-|-)}{C(+|-) - C(-|-) + C(-|+) - C(+|+)} \triangleq c \tag{2}$$

This demonstrates that, from a decision-theoretic perspective, the cost matrix has one degree of freedom. Without loss of generality, we can therefore assume that costs are expressed on an arbitrary (but linear) scale and that correct classifications have zero cost: under this interpretation, c quantifies the proportion of loss attributed to the negatives if equal numbers of positives and negatives are misclassified. This *relative cost* then gives the optimal threshold on the positive posterior. For example, if a false positive incurs four units of cost and a false negative one unit, then $c = 4/5$, and hence we would increase the decision threshold from the default, cost-indifferent threshold of $1/2$, in order to make fewer positive predictions which are much more costly when wrong.

How is the class posterior affected when the class prior changes, but the class-conditional likelihoods $P(X|Y)$ stay the same? Suppose the proportion of positives changes from π to π', and let p denote the posterior probability under prior π, then Elkan (2001) derives the following expression for the posterior probability under prior π' (assuming the class-conditional likelihoods remain unchanged):

$$p' = \pi' \frac{p - p\pi}{\pi - p\pi + \pi' p - \pi\pi'}$$

Noting that the denominator can be rewritten to $(1 - \pi)\pi' p + \pi(1 - \pi')(1 - p)$ and switching to odds, we obtain an expression that can be rewritten as a product of odds:

$$\frac{p'}{1 - p'} = \frac{\pi'}{1 - \pi'} \frac{1 - \pi}{\pi} \frac{p}{1 - p}$$

This is best interpreted right to left. The rightmost term is the posterior odds under the original prior π; multiplying this with the reciprocal of the original prior odds gives the likelihood ratio, and multiplying this again with the new prior odds gives the desired posterior odds under the new prior π'. For example, if at training time we have balanced classes ($\pi = 1/2$), while at deployment time we have 20 % positives ($\pi' = 1/5$), then p' is adjusted downward accordingly. Conversely, the (cost-indifferent) deployment decision threshold $p' = 1/2$ corresponds to $p = 4/5$, highlighting the duality with the cost-sensitive example above.

More generally, for $p' = 1/2$ we have that

$$p = \frac{(1 - \pi')\pi}{(1 - \pi')\pi + \pi'(1 - \pi)} \triangleq d \tag{3}$$

is the decision threshold on the original posterior that takes account of the changed class distribution. Hence, d parameterizes the distribution change from π to π' in the same way as c parameterizes the change from cost-indifference to cost-sensitivity. Clearly, this opens up the way to combining changes in both class and cost distribution in a straightforward way.

Evaluation Metrics for Calibration

A multiclass scoring classifier outputs, for every test instance x, a probability vector $(p_1(x), \ldots, p_k(x))$, where k is the number of classes and $\sum_{i=1}^{k} p_i(x) = 1$. Suppose the true class is represented by a bit vector $(b_1(x), \ldots, b_k(x))$ such that the bit corresponding to the true class is set to 1 and the remaining are 0, then the *Brier score* over a test set T is defined as

$$BS = \frac{1}{|T|} \sum_{x \in T} \frac{1}{2} \sum_{i=1}^{k} (p_i(x) - b_i(x))^2$$

The factor 1/2 ensures that the squared error per example is normalized between 0 and 1: the worst possible situation is that a wrong class is predicted with certainty, in which case two "bits" are wrong (Brier did not include this factor $1/2$ in his original account; Brier 1950). For binary classification this expression can be simplified to $BS = 1/|T| \sum_{x \in T}(p(x) - b(x))^2$, where $p(x)$ is the predicted probability of a designated class (the true class, say) and $b(x)$ is 1 if the designated class is the actual one and 0 otherwise.

Suppose we want to assign the same probability vector $(p_1, \ldots, p_k)$ to all labeled instances in a given set S – for example, all training instances that get filtered into the same leaf of a decision tree. Which assignment results in the lowest Brier score? As it turns out, this is exactly the empirical class distribution in the set, which will be denoted $(\dot{p}_1, \ldots, \dot{p}_k)$. This follows from the fact that the Brier score over S can be decomposed as

$$BS(S) = \frac{1}{2}\sum_{i=1}^{k}(p_i - \dot{p}_i)^2 + \frac{1}{2}\sum_{i=1}^{k}\dot{p}_i(1 - \dot{p}_i)$$

The first term in this decomposition is known as the *calibration loss*, and the second term is called the *refinement loss*. As both terms in this decomposition are nonnegative and refinement loss is independent of p_i, the overall expression is minimized by minimizing calibration loss, which gives $p_i = \dot{p}_i$ for all i.

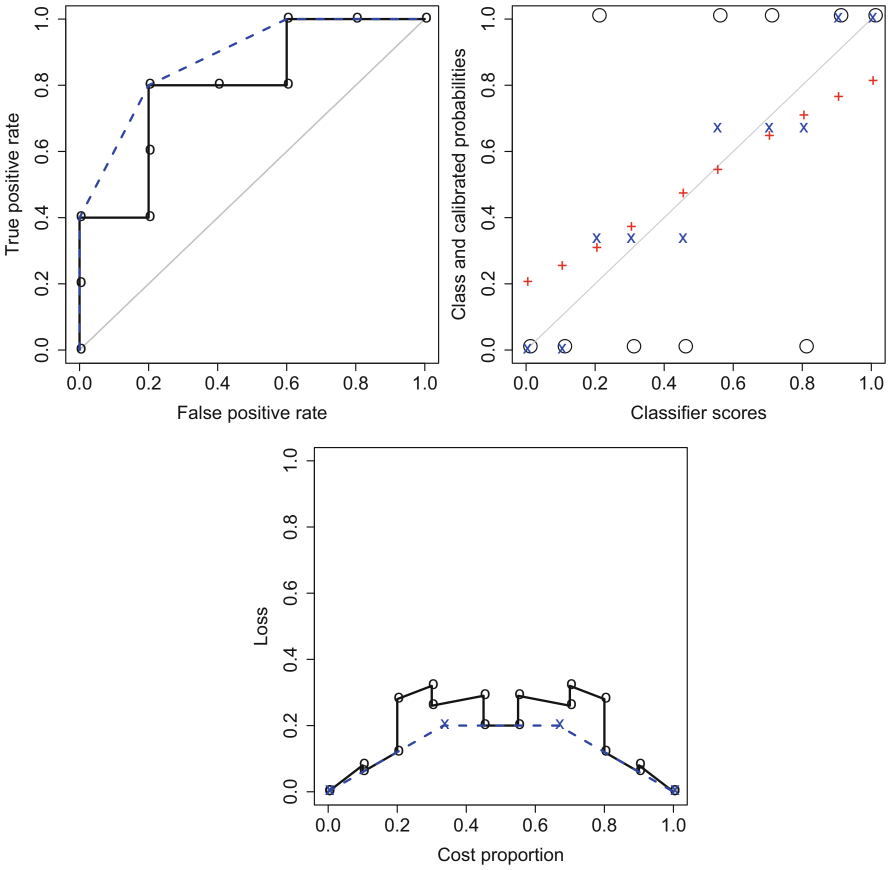
By taking a weighted average over all leaves of a decision tree, the decomposition can be applied to the Brier score over the entire data set. However, for a linear classifier which potentially assigns unique probabilities to every test instance, we need some way to group instances with similar probabilities together so that we can calculate the empirical probabilities. There are several ways of achieving this grouping, but the decomposition will in general be approximate (unless we have access to the true distributions, for which Hernández-Orallo et al. (2012) give the exact decomposition). Nevertheless, the important point to note is that Brier score is a combined measure of how well calibrated a classifier is and how well separated the scores for positives and negatives are.

The fact that Brier score is minimized if the predicted probabilities match the empirical probabilities identifies it as a so-called proper scoring rule (Gneiting and Raftery 2007). Many other proper scoring rules exist, including logarithmic loss which penalizes a probability vector with the negative logarithm of the probability assigned to the true class. These other scoring rules are amenable to similar decompositions: e.g., logarithmic loss uses Kullback-Leibler divergence between predicted and empirical probabilities to quantify calibration loss and Shannon entropy to quantify refinement loss of the empirical probabilities. Kull and Flach (2015) give an overview and also provides an underlying four-way decomposition which includes a model-independent irreducible component as well as a component that can be reduced to zero by a simple affine transformation of the scores.

Calibration Methods

In order to understand what it means for a classifier to be calibrated, it is instructive to consider its ROC curve (see ▶ ROC Analysis). A ROC curve plots true positive rate against false positive rate when the decision threshold is varied, and its slope is proportional to the empirical probability among instances receiving the same (or similar) scores from the classifier. Consider a small example in which a classifier assigns scores (1.00, 0.90, 0.80, 0.70, 0.55, 0.45, 0.30, 0.20, 0.10, 0.0) to 10 test instances with true classes $(1, 1, 0, 1, 1, 0, 0, 1, 0, 0)$, which is visualized in Fig. 1. The first thing we note is that the ROC curve on the top left has "dents" or concavities where the slope – and therefore the empirical probability – increases but the scores decrease (e.g., scores $(0.80, 0.70, 0.55)$ with empirical probabilities $(0, 1, 1)$). We can "repair" these concavities by tying the scores $(2/3, 2/3, 2/3)$, thereby forming the ROC convex hull (dashed line in the top left figure) and a piecewise constant calibration map (crosses in the middle figure).

The idea of using the ROC convex hull as a calibration method is related to isotonic regression, which differs from standard least-



Classifier Calibration, Fig. 1 (*top left*) Example ROC curve (*black*, *solid line*) and convex hull (*blue*, *dashed line*) on a small data set with 10 instances. (*top right*) Uncalibrated scores against true classes (*black circles*), calibrated scores obtained with isotonic regression (*blue crosses*), and logistic calibration (*red plusses*); the effect of calibration is larger for points further away from the diagonal. (*bottom*) Cost curves obtained when thresholding at cost proportion *c* for the original, uncalibrated scores (*black*, *solid line*) and isotonically calibrated scores (*blue*, *dashed line*); the difference between the two curves represents the decrease in Brier score achievable by calibration

squares regression in that the fitted line be piecewise constant. The standard algorithm to perform isotonic regression is the *pool adjacent violators* (PAV) method, which was introduced to the machine learning community by Zadrozny and Elkan (2001). The method, which we will call isotonic calibration in this entry, was demonstrated to be equivalent to the ROC convex hull algorithm by Fawcett and Niculescu-Mizil (2007). The key idea is that the slope of a ROC curve segment represents an empirical likelihood ratio *LR* and hence

$$p = \frac{LR}{LR + a} \tag{4}$$

with $a = (1 - \pi)/\pi$ is the corresponding empirical posterior probability (where π is the proportion of positives). See ▶ ROC Analysis for further details.

Alternatively, we can obtain *LR* from a parametric model. For example, suppose that the scores are obtained from a linear model $s(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} - t$ and assume for simplicity that the weight vector $\mathbf{w}$ is unit length, then $s(\mathbf{x})$ gives the signed distance of $\mathbf{x}$ from the decision boundary in the direction given by $\mathbf{w}$. The parametric assumption is that these distances are normally distributed within each class with the same variance σ^2, from which we can derive

$$
\begin{aligned}
LR(\mathbf{x}) &= \exp(\gamma(\mathbf{w}\cdot\mathbf{x} - t')) \\
&= \exp(\gamma(s(\mathbf{x}) - (t' - t))) \\
\gamma &= \mathbf{w}\cdot(\mu^+ - \mu^-)/\sigma^2 \\
t' &= \mathbf{w}\cdot(\mu^+ + \mu^-)/2
\end{aligned}
$$

where μ^+ and μ^- are the class means. Plugging this back into Eq. (4) and assuming $a = 1$ for simplicity gives

$$
\begin{aligned}
p(\mathbf{x}) &= \frac{1}{1 + \exp(-\gamma(\mathbf{w}\cdot\mathbf{x} - t'))} \\
&= \frac{1}{1 + \exp(-\gamma(s(\mathbf{x}) - (t' - t)))}
\end{aligned}
$$

Interpreted as a mapping from s to p, this defines a logistic curve with midpoint $s = t' - t$ and slope γ at this midpoint. The location parameter $t' - t$ shifts the decision boundary such that it cuts the line connecting the two class means halfway; the shape parameter γ quantifies how well separated the two classes are. This method was popularized by John Platt as a way to get probabilities out of ▸ support vector machines (Platt 2000) and is often referred to as *Platt scaling*; we call it logistic calibration in this entry.

Figure 1 shows the results of both logistic and isotonic calibration on the running example. The isotonically calibrated classifier both has higher AUC than the uncalibrated one (0.88 rather than 0.80) – since tying the scores as suggested by the ROC convex hull leads to a better ranking – and also lower Brier score (0.1333 rather than 0.1885). The latter is visualized in the cost curves on the bottom, which plot the cost-sensitive loss for different values of the cost proportion c (Drummond and Holte 2006). Treating the uncalibrated scores as if they were calibrated and hence using c for the decision threshold as derived in Eq. (2) yields the upper, solid cost curve, the area under which is equal to the uncalibrated Brier score (Hernández-Orallo et al. 2011). Using the isotonically calibrated scores yields the lower, dashed cost curve upon which we cannot improve without changing the model.

Also shown in the top right plot are the logistically calibrated scores; as this is a monotonic transformation, it doesn't affect the ranking or the ROC curve. The model assumptions of logistic calibration (normally distributed scores per class) are not really satisfied on this small example, and hence the logistically calibrated scores only lead to a small decrease in Brier score (0.1871) with a cost curve similar to the uncalibrated one (not shown).

If we decompose the Brier score into calibration loss and refinement loss, we see that for the uncalibrated scores refinement loss is zero as no instances from different classes receive the same score, so the Brier score equals calibration loss. Conversely, we see that for the isotonically calibrated scores, the calibration loss is zero, and hence the Brier score equals the refinement loss (on the labeled data on which calibration was carried out). Hence, isotonic calibration increases the refinement loss in order to achieve a larger decrease in calibration loss. In contrast, logistic calibration only affects calibration loss.

Practically speaking, calibration methods require a separate calibration set to avoid overfitting. For experimental studies on classifier calibration methods, see Niculescu-Mizil and Caruana (2005) and Bella et al. (2013).

Future Directions

Multiclass and Multilabel Calibration

Calibration of a multiclass classifier is not a solved problem, but there are several possible strategies. As logistic calibration essentially involves fitting a univariate logistic regression model to the scores output by the classifier, a natural way to extend this to more than two classes is to use multinomial logistic regression. This is again a parametric model which assumes that scores are normally distributed within each class.

A simple method that is recommended by Zadrozny and Elkan (2002) is to perform a logistic or isotonic calibration for each class separately, treating all other classes as the negative class (one-versus-rest). Since the resulting probabilities don't necessarily add up to one, they can be renormalized by dividing them by their sum.

Alternatively, one might consider to calibrate each class against each other class (one-versus-one). This results in more probabilities than there are classes, and hence methods to produce a multinomial probability vector are more involved. Hastie and Tibshirani (1998) proposed a solution called coupling, and Zadrozny (2001) generalized it to other code matrices (see ▶ Error Correcting Output Codes). Kong and Dietterich (1997) proposed an alternative method, also based on ECOC.

Multilabel classifiers differ from multiclass classifiers in that several labels may apply simultaneously. In the presence of sparse labels, calibration is particularly important, but it needs to be considered in the context of how multilabel classification performance will be evaluated. For example, one popular evaluation metric is Hamming loss which calculates the proportion of labels that are mispredicted – this effectively puts all labels in the same bag, and hence there is no point in separately calibrating the scores for each label. Other evaluation metrics are calculated label-wise or even instance-wise. Further research is needed to identify the right calibration methods for these cases.

Calibrating for Different Losses

Throughout this entry we have made an implicit assumption that our goal is to maximize accuracy: the proportion of correctly classified instances (true positives and true negatives). This justifies the additive cost model of Eq. (1) which led to the model-independent thresholds summarized in Eqs. (2) and (3). However, there is a range of recent results demonstrating that this threshold is suboptimal for other performance metrics. For example, Zhao et al. (2013) proved that if our goal is to maximize the F-score (the harmonic mean of precision and recall), then the optimal threshold θ^* on the true posterior is half the F-score obtained at that threshold – it follows that the optimal threshold for F-score is less than or equal to the optimal threshold for accuracy, with equality obtained only for perfect classifiers. Koyejo et al. (2014) extends the analysis to a family of performance metrics including accuracy and the F_β-score (a weighted harmonic mean of precision and recall, with $\beta = 1$ yielding the F-score). Specifically, if F^*_β denotes the optimal F_β-score achievable by a model, then $\theta^* = F^*_\beta/(1+\beta^2)$.

Instead of investigating how to adapt the decision rule on the class posterior to account for a different performance metric – which is necessarily classifier dependent – Flach and Kull (2015) suggest for the classifier to output a different score specifically adapted for the F_β metric. Let p be the calibrated posterior probability for a given instance; let B be the value of β for which the instance is on the F_β decision boundary (i.e., the F_β-score would be the same regardless of whether the instance is predicted to be positive or negative); and let F^*_B be the optimal F_β-score achievable by the model for that value of $\beta = B$; then the model outputs the score $p/F^*_B = 1/(1+B^2)$. All quantities involved can be precomputed from the ROC convex hull. This naturally leads to the idea of a classifier outputting *multiple* calibrated scores: calibrated estimates of the posterior probability for optimizing accuracy, adjustments of the posterior as just described for optimizing F-scores, and possibly others. The latter can still be seen as calibrated scores if we adopt a broader view of calibration: *a well-calibrated classifier calculates the cost parameters under which the expected cost for the instance under consideration is the same regardless of the predicted class.*

Cross-References

- ▶ Classification
- ▶ Class Imbalance Problem
- ▶ Cost-Sensitive Learning
- ▶ Logistic Regression
- ▶ Posterior Probability
- ▶ ROC Analysis

Recommended Reading

Bella A, Ferri C, Hernández-Orallo J, Ramírez-Quintana MJ (2013) On the effect of calibration in classifier combination. Appl Intell 38(4):566–585

Brier G (1950) Verification of forecasts expressed in terms of probabilities. Mon Weather Rev 78:1–3

Drummond C, Holte R (2006) Cost curves: an improved method for visualizing classifier performance. Mach Learn 65(1):95–130

Elkan C (2001) The foundations of cost-sensitive learning. In: Proceedings of 17th international joint conference on artificial intelligence (IJCAI'01). Morgan Kaufmann, pp 973–978

Fawcett T, Niculescu-Mizil A (2007) PAV and the ROC convex hull. Mach Learn 68(1):97–106

Ferri C, Flach P, Hernández-Orallo J (2003) Improving the AUC of probabilistic estimation trees. In: 14th European conference on machine learning (ECML'03). Springer, pp 121–132

Flach P, Kull M (2015) Precision-recall-gain curves: PR analysis done right. In: Advances in neural information processing systems (NIPS'15), pp 838–846

Gneiting T, Raftery AE (2007) Strictly proper scoring rules, prediction, and estimation. J Am Stat Assoc 102(477):359–378

Hastie T, Tibshirani R (1998) Classification by pairwise coupling. Ann Stat 26(2):451–471

Hernández-Orallo J, Flach P, Ferri C (2011) Brier curves: a new cost-based visualisation of classifier performance. In: Proceedings 28th international conference on machine learning (ICML'11), pp 585–592

Hernández-Orallo J, Flach P, Ferri C (2012) A unified view of performance metrics: translating threshold choice into expected classification loss. J Mach Learn Res 13(1):2813–2869

Kong EB, Dietterich T (1997) Probability estimation via error-correcting output coding. In: International conference on artificial intelligence and soft computing

Koyejo OO, Natarajan N, Ravikumar PK, Dhillon IS (2014) Consistent binary classification with generalized performance metrics. In: Advances in neural information processing systems (NIPS'14), pp 2744–2752

Kull M, Flach P (2015) Novel decompositions of proper scoring rules for classification: score adjustment as precursor to calibration. In: Machine learning and knowledge discovery in databases (ECML-PKDD'15). Springer, pp 68–85

Niculescu-Mizil A, Caruana R (2005) Predicting good probabilities with supervised learning. In: Proceedings of 22nd international conference on machine learning (ICML'05), pp 625–632

Platt J (2000) Probabilities for SV machines. In: Smola A, Bartlett P, Schölkopf B, Schuurmans D (eds) Advances in large margin classifiers. MIT Press, Cambridge, pp 61–74

Provost F, Domingos P (2003) Tree induction for probability-based ranking. Mach Learn 52(3):199–215

Zadrozny B (2001) Reducing multiclass to binary by coupling probability estimates. In: Advances in neural information processing systems (NIPS'01), pp 1041–1048

Zadrozny B, Elkan C (2001) Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In: Proceedings of 18th international conference on machine learning (ICML'01), pp 609–616

Zadrozny B, Elkan C (2002) Transforming classifier scores into accurate multiclass probability estimates. In: Proceedings of 8th international conference on knowledge discovery and data mining (KDD'02). ACM, pp 694–699

Zhao M-J, Edakunni N, Pocock A, Brown G (2013) Beyond Fano's inequality: bounds on the optimal F-score, BER, and cost-sensitive risk and their implications. J Mach Learn Res 14(1):1033–1090

Classifier Systems

Pier Luca Lanzi
Politecnico di Milano, Milano, Italy

Synonyms

Genetics-based machine learning; Learning classifier systems

Definition

Classifier systems are rule-based systems that combine ▶ temporal difference learning or ▶ supervised learning with a genetic algorithm to solve classification and ▶ reinforcement learning problems. Classifier systems come in two flavors: Michigan classifier systems, which are designed for online learning, but can also tackle offline problems; and Pittsburgh classifier systems, which can only be applied to offline learning.

In Michigan classifier systems (Holland 1976), learning is viewed as an online adaptation process to an unknown environment that represents the problem and provides feedback in terms of a numerical reward. Michigan classifier systems maintain a single candidate solution consisting of a set of rules, or a population of classifiers. Michigan systems apply (1) temporal difference learning to distribute the incoming reward to the classifiers that are accountable for it; and (2) a genetic algorithm to select, recombine, and mutate individual classifiers so as to improve their contribution to the current solution.

In contrast, in Pittsburgh classifier systems (Smith 1980), learning is viewed as an offline optimization process in which a genetic algorithm alone is applied to search for the best solution to a given problem. In addition, Pittsburgh classifier systems maintain not one, but a set of candidate solutions. While in the Michigan classifier system each individual classifier represents a part of the overall solution, in the Pittsburgh system each individual is a complete candidate solution (itself consisting of a set of classifiers). The fitness of each Pittsburgh individual is computed offline by testing it on a representative sample of problem instances. The individuals compete among themselves through selection, while crossover and mutation recombine solutions to search for better solutions.

Motivation and Background

Machine learning is usually viewed as a search process in which a solution space is explored until an appropriate solution to the target problem is found (Mitchell 1982) (see ▸ Supervised Learning). Machine learning methods are characterized by the way they represent solutions (e.g., using ▸ decision trees, rules), by the way they evaluate solutions (e.g., classification accuracy, information gain) and by the way they explore the solution space (e.g., using a ▸ general-to-specific strategy or a specific-to-general strategy).

Classifier systems are methods of *genetics-based* machine learning introduced by Holland, the father of ▸ genetic algorithms. They made their first appearance in Holland (1976) where the first diagram of a classifier system, labeled "cognitive system," was shown. Subsequently, they were described in detail in the paper "Cognitive Systems based on Adaptive Algorithms" (Holland and Reitman 1978). Classifier systems are characterized by a rule-based representation of solutions and a genetics-based exploration of the solution space. While other ▸ rule learning methods, such as CN2 (Clark and Niblett 1989) and FOIL (Quinlan and Cameron-Jones 1995), generate one rule at a time following a sequential covering strategy (see ▸ Covering Algorithm), classifier systems work on one or more solutions at once, and they explore the solution space by applying the principles of natural selection and genetics.

In classifier systems (Holland 1976; Holland and Reitman 1978; Wilson 1995), machine learning is modeled as an online adaptation process to an unknown *environment*, which provides feedback in terms of a numerical reward. A classifier system perceives the environment through its detectors and, based on its sensations, it selects an action to be performed in the environment through its effectors. Depending on the efficacy of its actions, the environment may eventually reward the system. A classifier system learns by trying to maximize the amount of reward it receives from the environment. To pursue such a goal, it maintains a set (a *population*) of condition-action-prediction rules, called *classifiers*, which represents the current solution. Each classifier's condition identifies some part of the problem domain; the classifier's action represents a decision on the subproblem identified by its condition; and the classifier's prediction, or strength, estimates the value of the action in terms of future rewards on that subproblem. Two separate components, credit assignment and rule discovery, act on the population with different goals. ▸ Credit assignment, implemented either by methods of temporal difference or supervised learning, exploits the incoming reward to estimate the action values in each subproblem so as to identify the best classifiers in the population. At the same time, rule discovery, usually implemented by a genetic algorithm, selects, recombines, and mutates the classifiers in the population to improve the current solution.

Classifier systems were initially conceived as modeling tools. Given a real system with unknown underlying dynamics, for instance a financial market, a classifier system would be used to generate a behavior that matched the real system. The evolved rules would provide a plausible, human readable model of the unknown system – a way to look inside the box. Subsequently, with the developments in the area of machine learning and the rise of reinforcement learning, classifier systems have been more and more often studied and presented as alternatives to other machine learning methods. Wilson's XCS (1995), the most successful classifier system to date, has proven to be both a valid alternative to other reinforcement learning approaches and an effective approach to classification and data mining (Bull 2004; Bull and Kovacs 2005; Lanzi et al. 2000).

Kenneth de Jong and his students (de Jong 1988; Smith 1980, 1983) took a different perspective on genetics-based machine learning and modeled learning as an *optimization* process rather than an *adaptation* process as done in Holland (1976). In this case, the solution space is explored by applying a genetic algorithm to a population of individuals each representing a *complete* candidate solution – that is, a set of rules (or a production system, de Jong 1988; Smith 1980. At each cycle, a critic is applied to each individual (to each set of rules) to obtain a performance measure that is then used by the genetic algorithm to guide the exploration of the solution space. The individuals in the population compete among themselves through selection, while crossover and mutation recombine solutions to search for better ones.

The approaches of Holland (Holland 1976; Holland and Reitman 1978) and de Jong (de Jong 1988; Smith 1980, 1983) have been extended and improved in several ways (see Lanzi et al. (2000) for a review). The models of classifier systems that are inspired by the work of Holland (1976) at the University of Michigan are usually called Michigan classifier systems; the ones that are inspired by Smith (1980, 1983) and de Jong (1988) at the University of Pittsburgh are usually termed Pittsburgh classifier systems – or briefly, Pitt classifier systems.

Pittsburgh classifier systems separate the evaluation of candidate solutions, performed by an external critic, from the genetic search. As they evaluate candidate solutions as a whole, Pittsburgh classifier systems can easily identify and emphasize sequentially cooperating classifiers, which is particularly helpful in problems involving partial observability. In contrast, in Michigan classifier systems the credit assignment is focused, due to identification of the actual classifiers that produce the reward, so learning is *much* faster but sequentially cooperating classifiers are more difficult to spot. As Pittsburgh classifier systems apply the genetic algorithm to a set of solutions, they only work offline, whereas Michigan classifier systems work online, although they can also tackle offline problems. Finally, the design of Pittsburgh classifier systems involves decisions as to how an entire solution should be represented and how solutions should be recombined – a task which can be daunting. In contrast, the design of Michigan classifier systems involves simpler decisions about how a rule should be represented and how two rules should be recombined. Accordingly, while the representation of solutions and its related issues play a key role in Pittsburgh models, Michigan models easily work with several types of representations (Lanzi 2001; Lanzi and Perrucci 1999; Mellor 2005).

Structure of the Learning System

Michigan and Pittsburgh classifier systems were both inspired by the work of Holland on the broadcast language (Holland 1975). However, their structures reflect two different ways to model machine learning: as an adaptation process in the case of Michigan classifier systems; and as an optimization problem, in the case of Pittsburgh classifier systems. Thus, the two models, originating from the same idea (Holland's broadcast language), have radically different structures.

Michigan Classifier Systems

Holland's classifier systems define a general paradigm for genetics-based machine learning. The description in Holland and Reitman (1978) provides a list of principles for online learning through adaptation. Over the years, such principles have guided researchers who developed several models of Michigan classifier systems (Butz 2002; Wilson 1994, 1995, 2002) and applied them to a large variety of domains

(Bull 2004; Lanzi and Riolo 2003; Lanzi et al. 2000). These models extended and improved Holland's original ideas, but kept all the ingredients of the original recipe: a population of classifiers, which represents the current system knowledge; a performance component, which is responsible for the short-term behavior of the system; a credit assignment (or reinforcement) component, which distributes the incoming reward among the classifiers; and a rule discovery component, which applies a genetic algorithm to the classifiers to improve the current knowledge.

Knowledge Representation

In Michigan classifier systems, knowledge is represented by a population of classifiers. Each classifier is usually defined by four main parameters: the *condition*, which identifies some part of the problem domain; the *action*, which represents a decision on the subproblem identified by its condition; the *prediction* or strength, which estimates the amount of reward that the system will receive if its action is performed; and finally, the *fitness*, which estimates how good the classifier is in terms of problem solution.

The knowledge representation of Michigan classifier systems is extremely flexible. Each one of the four classifier components can be tailored to fit the need of a particular application, without modifying the main structure of the system. In problems involving binary inputs, classifier conditions can be simply represented using strings defined over the alphabet {0, 1, #}, as done in Holland and Reitman (1978), Goldberg (1989), and Wilson (1995). In problems involving real inputs, conditions can be represented as disjunctions of intervals, similar to the ones produced by other rule learning methods (Clark and Niblett 1989). Conditions can also be represented as general-purpose symbolic expressions (Lanzi 2001; Lanzi and Perrucci 1999) or first-order logic expressions (Mellor 2005). Classifier actions are typically encoded by a set of symbols (either binary strings or simple labels), but continuous real-valued actions are also available (Wilson 2007). Classifier prediction (or strength) is usually encoded by a parameter (Goldberg 1989; Holland and Reitman 1978; Wilson 1995). However, classifier prediction can also be computed using a parameterized function (Wilson 2002), which results in solutions represented as an ensemble of local approximators – similar to the ones produced in generalized reinforcement learning (Sutton and Barto 1998).

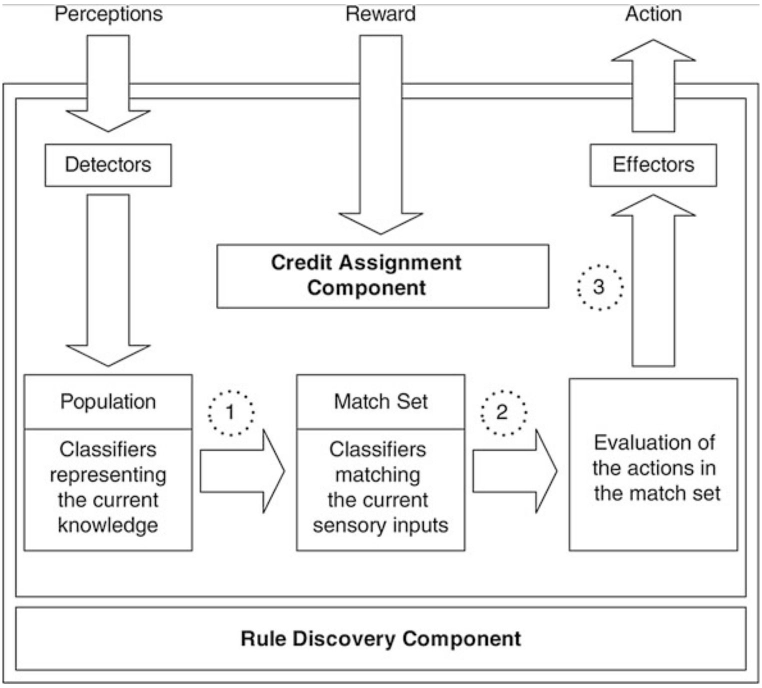
Performance Component

A simplified structure of Michigan classifier systems is shown in Fig. 1. We refer the reader to Goldberg (1989) and Holland and Reitman (1978) for a detailed description of the original model and to Butz (2002) and Wilson (1994, 1995, 2001) for descriptions of recent classifier system models.

A classifier system learns through trial and error interactions with an unknown environment. The system and the environment interact continually. At each time step, the classifier system perceives the environment through its detectors; it builds a *match set* containing all the classifiers in the population whose condition matches the current sensory input. The match set typically contains classifiers that advocate contrasting actions; accordingly, the classifier system evaluates each action in the match set, and selects an action to be performed balancing exploration and exploitation. The selected action is sent to the effectors to be executed in the environment; depending on the effect that the action has in the environment, the system receives a scalar reward.

Credit Assignment

The *credit assignmentcomponent* (also called reinforcement component, Wilson 1995) distributes the incoming reward to the classifiers that are accountable for it. In Holland and Reitman (1978), credit assignment is implemented by Holland's bucket brigade algorithm (Holland 1986), which was partially inspired by the credit allocation mechanism used by Samuel in his pioneering work on learning checkers-playing programs (Samuel 1959).


Perceptions
Reward
Action
Detectors
Effectors
Credit Assignment Component
3
Population
Classifiers representing the current knowledge
1
Match Set
Classifiers matching the current sensory inputs
2
Evaluation of the actions in the match set
Rule Discovery Component

Classifier Systems, Fig. 1 Simplified structure of a Michigan classifier system. The system perceives the environment through its detectors and (1) it builds the match set containing the classifiers in the population that match the current sensory inputs; then (2) all the actions in the match set are evaluated, and (3) an action is selected to be performed in the environment through the effectors

In the early years, classifier systems and the bucket brigade algorithm were confined to the evolutionary computation community. The rise of reinforcement learning increased the connection between classifier systems and temporal difference learning (Sutton 1988; Sutton and Barto 1998): in particular, Sutton (1988) showed that the bucket brigade algorithm is a kind of temporal difference learning, and similar connections were also made in Watkins (1989) and Dorigo and Bersini (1994). Later, the connection between classifier systems and reinforcement learning became tighter with the introduction of Wilson's XCS (1995), in which credit assignment is implemented by a modification of Watkins Q-learning (Watkins 1989). As a consequence, in recent years, classifier systems are often presented as methods of reinforcement learning with genetics-based generalization (Bull and Kovacs 2005).

Rule Discovery Component

The *rule discovery component* is usually implemented by a genetic algorithm that selects classifiers in the population with probability proportional to their fitness; it copies the selected classifiers and applies genetic operators (usually crossover and mutation) to the offspring classifiers; the new classifiers are inserted in the population, while other classifiers are deleted to keep the population size constant.

Classifiers selection plays a central role in rule discovery. Classifier selection depends on the definition of classifier fitness and on the subset of classifiers considered during the selection process. In Holland and Reitman (1978), classifier fitness coincides with classifier prediction, while selection is applied to all the classifiers in the population. This approach results in a pressure toward classifiers predicting high returns, but typically tends to produce overly general solutions. To avoid such solutions, Wilson (1995) introduced the XCS classifier system in which accuracy-based fitness is coupled with a niched genetic algorithm. This approach results in a pressure toward accurate maximally general classifiers, and has made XCS the most successful classifier system to date.

Pittsburgh Classifier Systems

The idea underlying the development of Pittsburgh classifier systems was to show that interesting behaviors could be evolved using a simpler model than the one proposed by Holland with Michigan classifier systems (Holland 1976; Holland and Reitman 1978).

In Pittsburgh classifier systems, each individual is a set of rules that encodes an entire candidate solution; each rule has a fixed length, but each rule set (each individual) usually contains a variable number of rules. The genetic operators, crossover and mutation, are tailored to the rule-based, variable-length representation. The individuals in the population compete among themselves, following the selection-recombination-mutation cycle that is typical of genetic algorithms (Goldberg 1989; Holland 1975). While in Michigan classifier systems individuals in the population (the single rules) cooperate, in Pittsburgh classifier systems there is no cooperation among individuals (the rule sets), so that the genetic algorithm operation is simpler for Pittsburgh models. However, as Pittsburgh classifier systems explore a much larger search space, they usually require more computational resources than Michigan classifier systems.

The pseudo-code of a Pittsburgh classifier system is shown in Fig. 2. At first, the individuals in the population are randomly initialized (line 2). At time t, the individuals are evaluated by an external critic, which returns a performance measure that the genetic algorithm exploits to compute the fitness of individuals (lines 3 and 10). Following this, selection (line 6), recombination, and mutation (line 7) are applied to the individuals in the population – as done in a typical genetic algorithm. The process stops when a termination criterion is met (line 4), usually when an appropriate solution is found.

The design of Pittsburgh classifier systems follows the typical steps of genetic algorithm design, which means deciding how a rule set should be represented, what genetic operators should be applied, and how the fitness of a set of rules should be calculated. In addition, Pittsburgh classifier systems need to address the *bloat* phenomenon (Tackett 1994) that arises with any variable-sized representation, like the rule sets evolved by Pittsburgh classifier systems. Bloat can be defined as the growth of individuals without an actual fitness improvement. In Pittsburgh classifier systems, bloat increases the size of candidate solutions by adding useless rules to individuals, and it is typically limited by introducing a parsimony pressure that discourages large rule sets (Bassett and de Jong 2000). Alternatively, Pittsburgh classifier systems can be combined with multi-objective optimization, so as to separate the maximization of the rule set performance and the minimization of the rule set size.

Examples of Pittsburgh classifier systems include SAMUEL (Grefenstette et al. 1990), the Genetic Algorithm Batch-Incremental Concept Learner (GABIL) (de Jong and Spears 1991), GIL Janikow (1993), GALE (Llorá 2002), and GAssist (Bacardit 2004).

Applications

Classifier systems have been applied to a large variety of domains, including computational

Classifier Systems, Fig. 2 Pseudo-code of a Pittsburgh classifier system

```
1.   t := 0
2.   Initialize the population P(t)
3.   Evaluate the rules sets in P(t)
4.   While the termination condition is not satisfied
5.   Begin
6.      Select the rule sets in P(t) and generate Ps(t)
7.      Recombine and mutate the rule sets in Ps(t)
8.      P(t+1) := Ps(t)
9.      t := t+1
10.     Evaluate the rules sets in P(t)
11.  End
```

economics (e.g., Arthur et al. 1996), autonomous robotics (e.g., Dorigo and Colombetti 1998), classification (e.g., Barry et al. 2004), fighter aircraft maneuvering (Bull 2004; Smith et al. 2000), and many others. Reviews of classifier system applications are available in Lanzi et al. (2000); Lanzi and Riolo (2003), and Bull (2004).

Programs and Data

The major sources of information about classifier systems are the LCSWeb maintained by Alwyn Barry, which can be reached through, and www.learning-classifier-systems.org maintained by Xavier Llorà.

Several implementations of classifier systems are freely available online. The first standard implementation of Holland's classifier system in Pascal was described in Goldberg (1989), and it is available at http://www.illigal.org/; a C version of the same implementation, developed by Robert E. Smith, is available at http://www.etsimo.uniovi.es/ftp/pub/EC/CFS/src/. Another implementation of an extension of Holland's classifier system in C by Rick L. Riolo is available at http://www.cscs.umich.edu/Software/Contents.html. Implementations of Wilson's XCS 1995 are distributed by Alwyn Barry at the LCSWeb, by Martin V. Butz (at www.illigal.org), and by Pier Luca Lanzi (at xcslib.sf.net). Among the implementations of Pittsburgh classifier systems, the Samuel system is available from Alan C. Schultz at http://www.nrl.navy.mil/; Xavier Llorà distributes GALE (Genetic and Artificial Life Environment) a fine-grained parallel genetic algorithm for data mining at www.illigal.org/xllora.

Cross-References

- ▸ Credit Assignment
- ▸ Genetic and Evolutionary Algorithms
- ▸ Reinforcement Learning
- ▸ Rule Learning

Recommended Reading

Arthur BW, Holland JH, LeBaron B, Palmer R, Talyer P (1996) Asset pricing under endogenous expectations in an artificial stock market. Technical report, Santa Fe Institute

Bacardit i Peñarroya J (2004) Pittsburgh genetic-based machine learning in the data mining era: representations, generalization, and run-time. PhD thesis, Computer Science Department, Enginyeria i Arquitectura La Salle Universitat Ramon Llull, Barcelona

Barry AM, Holmes J, Llora X (2004) Data mining using learning classifier systems. In: Bull L (ed) Applications of learning classifier systems, studies in fuzziness and soft computing, vol 150. Springer, Pagg, pp 15–67

Bassett JK, de Jong KA (2000) Evolving behaviors for cooperating agents. In: Proceedings of the twelfth international symposium on methodologies for intelligent systems. LNAI, vol 1932. Springer, Berlin

Booker LB (1989) Triggered rule discovery in classifier systems. In: Schaffer JD (ed) Proceedings of the 3rd international conference on genetic algorithms (ICGA89). Morgan Kaufmann, San Francisco

Bull L (ed) (2004) Applications of learning classifier systems, studies in fuzziness and soft computing, vol 150. Springer, Berlin. ISBN 978-3-540-21109-9

Bull L, Kovacs T (eds) (2005) Foundations of learning classifier systems, studies in fuzziness and soft computing, vol 183. Springer, Berlin. ISBN 978-3-540-25073-9

Butz MV (2002) Anticipatory learning classifier systems. Genetic algorithms and evolutionary computation. Kluwer, Boston Academic Publishers.

Clark P, Niblett T (1989) The CN2 induction algorithm. Mach Learn 3(4):261–283

de Jong K (1988) Learning with genetic algorithms: an overview. Mach Learn 3(2–3):121–138

de Jong KA, Spears WM (1991) Learning concept classification rules using genetic algorithms. In: Proceedings of the international joint conference on artificial intelligence. Morgan Kaufmann, San Francisco, pp 651–656

Dorigo M, Bersini H (1994) A comparison of Q-learning and classifier systems. In: Cliff D, Husbands P, Meyer J-A, Wilson SW (eds) From animals to animats 3: proceedings of the third international conference on simulation of adaptive behavior. MIT Press, Cambridge, pp 248–255

Dorigo M, Colombetti M (1998) Robot shaping: an experiment in behavior engineering. MIT Press/Bradford Books, Cambridge

Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading

Grefenstette JJ, Ramsey CL, Schultz A (1990) Learning sequential decision rules using simulation models and competition. Mach Learn 5(4):355–381

Holland J (1986) Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning, an artificial intelligence approach, vol II, Chap. 20. Morgan Kaufmann, San Francisco, pp 593–623

Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor (Reprinted by the MIT Press in 1992)

Holland JH (1976) Adaptation. Progress in theoretical biology 4:263–293

Holland JH, Reitman JS (1978) Cognitive systems based on adaptive algorithms. In: Waterman DA, Hayes-Roth F (eds) Pattern-directed inference systems. Academic Press, New York (Reprinted from Evolutionary computation. The fossil record. Fogel DB (ed.) IEEE Press (1998))

Janikow CZ (1993) A knowledge-intensive genetic algorithm for supervised learning. Mach Learn 13(2–3):189–228

Lanzi PL (2001) Mining interesting knowledge from data with the XCS classifier system. In: Spector L, Goodman ED, Wu A, Langdon WB, Voigt H-M, Gen M et al (eds) Proceedings of the genetic and evolutionary computation conference (GECCO-2001). Morgan Kaufmann, San Francisco, pp 958–965

Lanzi PL (2005) Learning classifier systems: a reinforcement learning perspective. In: Bull L, Kovacs T (eds) Foundations of learning classifier systems, studies in fuzziness and soft computing. Springer, Berlin, pp 267–284

Lanzi PL, Perrucci A (1999) Extending the representation of classifier conditions part II: from messy coding to S-expressions. In: Banzhaf W, Daida J, Eiben AE, Garzon MH, Honavar V, Jakiela M, Smith RE (eds) Proceedings of the genetic and evolutionary computation conference (GECCO 99). Morgan Kaufmann, Orlando, pp 345–352

Lanzi PL, Riolo RL (2003) Recent trends in learning classifier systems research. In: Ghosh A, Tsutsui S (eds) Advances in evolutionary computing: theory and applications. Springer, Berlin, pp 955–988

Lanzi PL, Stolzmann W, Wilson SW (eds) (2000) Learning classifier systems: from foundations to applications. Lecture notes in computer science, vol 1813. Springer, Berlin

Llorá X (2002) Genetics-based machine learning using fine-grained parallelism for data mining. PhD thesis, Enginyeria i Arquitectura La Salle, Ramon Llull University, Barcelona

Mellor D (2005) A first order logic classifier system. In: Beyer H (ed) Proceedings of the 2005 conference on genetic and evolutionary computation (GECCO '05). ACM Press, New York, pp 1819–1826

Quinlan JR, Cameron-Jones RM (1995) Induction of logic programs: FOIL and related systems. New Gener Comput 13(3–4):287–312

Samuel AL (1959) Some studies in machine learning using the game of checkers. In: Feigenbaum, Feldman J (eds) Computers and thought. McGraw-Hill, New York

Smith RE, Dike BA, Niehra RK, Ravichandran B, El-Fallah A (2000) Classifier systems in combat: two-sided learning of maneuvers for advanced fighter aircraft. Comput Methods Appl Mech Eng 186(2–4):421–437

Smith SF (1980) A learning system based on genetic adaptive algorithms. Doctoral dissertation, Department of Computer Science, University of Pittsburgh

Smith SF (1983) Flexible learning of problem solving heuristics through adaptive search. In: Proceedings of the eighth international joint conference on artificial intelligence. Morgan Kaufmann, Los Altos, pp 421–425

Sutton RS (1988) Learning to predict by the methods of temporal differences. Mach Learn 3:9–44

Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT Press, Cambridge

Tackett WA (1994) Recombination, selection, and the genetic construction of computer programs. Unpublished doctoral dissertation, University of Southern California

Watkins C (1989) Learning from delayed rewards. PhD thesis, King's College

Wilson SW (1995) Classifier fitness based on accuracy. Evol Comput 3(2):149–175

Wilson SW (2002) Classifiers that approximate functions. Natl Comput 1(2–3):211–234

Wilson SW (2007). "Three architectures for continuous action" learning classifier systems. International workshops, IWLCS 2003–2005, revised selected papers. In: Kovacs T, Llorà X, Takadama K, Lanzi PL, Stolzmann W, Wilson SW (eds) Lecture notes in artificial intelligence, vol 4399. Springer, Berlin, pp 239–257

Clause

A *clause* is a logical rule in a ▶ logic program. Formally, a clause is a disjunction of (possibly negated) literals, such as

$$\mathit{grandfather}(x, y) \vee \neg \mathit{father}(x, z) \vee \neg \mathit{parent}(z, y)$$

In the logic programming language ▶ Prolog this clause is written as

```
grandfather(X,Y) :- father(X,Z),
                    parent(Z,Y).
```

The part to the left of :- ("if") is the *head* of the clause, and the right part is its *body*. Informally, the clause asserts the truth of the head given the truth of the body. A clause with exactly one literal in the head is called a *Horn clause* or *definite clause*; logic programs mostly consist of definite clauses. A clause without a body is also called a *fact*; a clause without a head is also called a *denial*, or a *query* in a proof by refutation. The clause without head or body is called the *empty clause*: it signifies inconsistency or falsehood and is denoted □. Given a set of clauses, the *resolution* inference rule can be used to deduce logical consequences and answer queries (see ▶ First-Order Logic).

In machine learning, clauses can be used to express classification rules for structured individuals. For example, the following definite clause classifies a molecular compound as carcinogenic if it contains a hydrogen atom with charge above a certain threshold.

```
carcinogenic(M) :- atom(M,A1),
                   element(A1,h),
                   charge(A1,C1),
                   geq(C1,0.168).
```

Cross-References

▶ First-Order Logic
▶ Inductive Logic Programming
▶ Learning from Structured Data
▶ Logic Program
▶ Prolog

Clause Learning

In ▶ speedup learning, clause learning is a ▶ deductive learning technique used for the purpose of ▶ intelligent backtracking in satisfiability solvers. The approach analyzes failures at backtracking points and derives clauses that must be satisfied by the solution. The clauses are added to the set of clauses from the original satisfiability problem and serve to prune new search nodes that violate them.

Click-Through Rate (CTR)

CTR measures the success of a ranking of search results, or advertisement placing. Given the number of *impressions*, the number of times a web result or ad has been displayed, and the number of *clicks*, the number of users who clicked on the result/advertisement, CTR is the number of clicks divided by the number of impressions.

Clonal Selection

The clonal selection theory (CST) is the theory used to explain the basic response of the adaptive immune system to an antigenic stimulus. It establishes the idea that only those cells capable of recognizing an antigenic stimulus will proliferate, thus being selected against those that do not. Clonal selection operates on both T-cells and B-cells. When antibodies on a B-cell bind with an antigen, the B-cell becomes activated and begins to proliferate. New B-cell clones are produced that are an exact copy of the parent B-cell, but then they undergo somatic hypermutation and produce antibodies that are specific to the invading antigen. The B-cells, in addition to proliferating or differentiating into *plasma cells*, can differentiate into long-lived B *memory cells*. Plasma cells produce large amounts of *antibody* which will attach themselves to the antigen and act as a type of *tag* for T-cells to pick up on and remove from the system. This whole process is known as *affinity maturation*. This process forms the basis of many artificial immune system algorithms such as AIRS and aiNET.

Closest Point

▶ Nearest Neighbor

Cluster Editing

The Cluster Editing problem is almost equivalent to Correlation Clustering on complete instances. The idea is to obtain a graph that consists only of cliques. Although Cluster Deletion requires us to delete the smallest number of edges to obtain such a graph, in Cluster Editing we are permitted to add as well as remove edges. The final variant is Cluster Completion in which edges can only be added: each of these problems can be restricted to building a specified number of cliques.

Cluster Ensembles

Cluster ensembles are an unsupervised ▶ ensemble learning method. The principle is to create multiple different clusterings of a dataset, possibly using different algorithms, then aggregate the opinions of the different clusterings into an ensemble result. The final ensemble clustering should be in theory more reliable than the individual clusterings.

Cluster Initialization

▶ *K*-Means Clustering

Cluster Optimization

▶ Evolutionary Clustering

Clustering

Clustering is a type of ▶ unsupervised learning in which the goal is to partition a set of ▶ examples into groups called clusters. Intuitively, the examples within a cluster are more similar to each other than to examples from other clusters. In order to measure the similarity between examples, clustering algorithms use various distortion or ▶ distance measures. There are two major types clustering approaches: generative and discriminative. The former assumes a parametric form of the data and tries to find the model parameters that maximize the probability that the data was generated by the chosen model. The latter represents graph-theoretic approaches that compute a similarity matrix defined over the input data.

Cross-References

- ▶ Categorical Data Clustering
- ▶ Cluster Editing
- ▶ Cluster Ensembles
- ▶ Clustering
- ▶ Clustering from Data Streams
- ▶ Consensus Clustering
- ▶ Constrained Clustering
- ▶ Correlation Clustering
- ▶ Cross-Language Document Categorization
- ▶ Density-Based Clustering
- ▶ Dirichlet Process
- ▶ Evolutionary Clustering
- ▶ Graph Clustering
- ▶ *K*-Means Clustering
- ▶ *K*-Mediods Clustering
- ▶ Model-Based Clustering
- ▶ Partitional Clustering
- ▶ Projective Clustering
- ▶ Sublinear Clustering

Clustering Aggregation

▶ Consensus Clustering

Clustering Ensembles

▶ Consensus Clustering

Clustering from Data Streams

João Gama
University of Porto, Porto, Portugal

Abstract

Clustering is one of the most popular data mining techniques. In this article, we review the

relevant methods and algorithms for designing cluster algorithms under the data streams computational model, and discuss research directions in tracking evolving clusters.

Definition

Clustering is the process of grouping objects into different groups, such that the common properties of data in each subset are high and between different subsets are low. The data stream clustering problem is defined as *to maintain a continuously consistent good clustering of the sequence observed so far, using a small amount of memory and time*. The issues are imposed by the continuous arriving data points and the need to analyze them in real time. These characteristics require incremental clustering, maintaining cluster structures that evolve over time. Moreover, the data stream may evolve over time, and new clusters might appear, other disappears, reflecting the dynamics of the stream.

Main Techniques

Clustering data streams requires a process able to continuously cluster objects within memory and time restrictions (Gama 2010). Following Silva et al. (2013), algorithms for clustering data streams should ideally fulfill the following requirements:

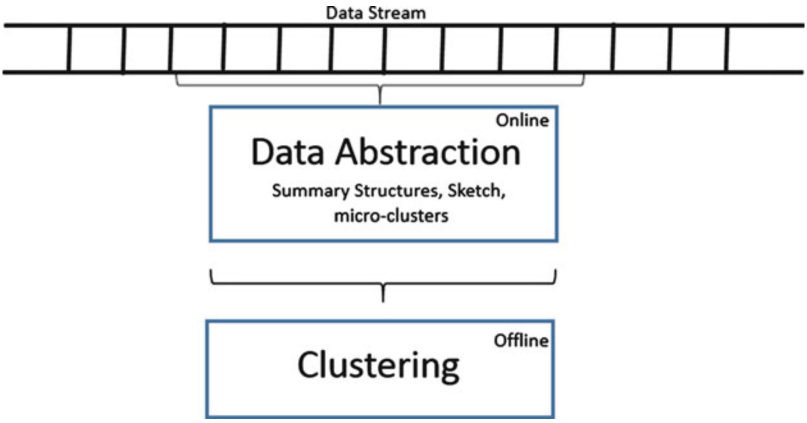
(i) provide timely results by performing fast and incremental processing of data objects;
(ii) rapidly adapt to changing dynamics of the data, which means algorithms should detect when new clusters may appear or others disappear;
(iii) scale to the number of objects that are continuously arriving;
(iv) provide a model representation that is not only compact, but that also does not grow with the number of objects processed (notice that even a linear growth should not be tolerated);
(v) rapidly detect the presence of outliers and act accordingly; and
(vi) deal with different data types, e.g., XML trees, DNA sequences, and GPS temporal and spatial information. Although these requirements are only partially fulfilled in practice, it is instructive to keep them in mind when designing algorithms for clustering data streams.

Major clustering approaches in data stream cluster analysis include:

- *Partitioning* algorithms: construct a partition of a set of objects into k clusters, which minimize some objective function (e.g., the sum of squared distances to the centroid representative). Examples include k-means (Farnstrom et al. 2000) and k-medoids (Guha et al. 2003);
- *Micro-clustering* algorithms: divide the clustering process into two phases, where the first phase is online and summarizes the data stream in local models (micro-clusters) and the second phase generates a global cluster model from the micro-clusters. Examples of these algorithms include BIRCH (Zhang et al. 1996), CluStream (Aggarwal et al. 2003), and ClusTree (Kranen et al. 2011).

Basic Concepts

Data stream clustering algorithms can be summarized into two main steps: data summarization step and clustering step, as illustrated in Fig. 1. The online abstraction step summarizes the data stream with the help of particular data structures in order to deal with space and memory constraints of stream applications. These data structures summarize the stream in order to preserve the meaning of the original objects without the need of storing them. Among the commonly employed data structures, we highlight the feature vectors (Zhang et al. 1996; Aggarwal et al. 2003), prototype arrays (Guha et al. 2003), coreset trees (Ackermann et al. 2012), and data grids (Gama et al. 2011).



Clustering from Data Streams, Fig. 1 A generic schema for clustering data streams

A powerful idea in clustering from data streams is the concept of *cluster feature – CF*. A cluster feature, or *micro-cluster*, is a compact representation of a set of points. A CF structure is a triple (N, LS, SS), used to store the sufficient statistics of a set of points:

- N is the number of data points;
- LS is a vector, of the same dimension of data points, that store the linear sum of the N points;
- SS is a vector, of the same dimension of data points, that store the square sum of the N points.

The properties of cluster features are:

- **Incrementality**

  If a point x is added to a cluster A, the sufficient statistics are updated as follows:

  $$LS_A \leftarrow LS_A + x; SS_A \leftarrow SS_A + x^2; N_A \leftarrow N_A + 1$$

- **Additivity**

  If A and B are disjoint sets, merging them is equal to the sum of their parts. The additive property allows us to merge subclusters incrementally:

  $$LS_C \leftarrow LS_A + LS_B; SS_C \leftarrow SS_A + SS_B; N_C \leftarrow N_A + N_B.$$

A CF entry has sufficient information to calculate the norms

$$L_1 = \sum_{i=1}^{n} |LS_{a_i} - LS_{b_i}| \text{ and}$$

$$L_2 = \sqrt{\sum_{i=1}^{n} (LS_{a_i} - LS_{b_i})^2}$$

and basic measures to characterize a cluster:

- **Centroid**, defined as the gravity center of the cluster:

  $$\vec{X}0 = \frac{LS}{N}$$

- **Radius**, defined as the average distance from member points to the centroid:

  $$R = \sqrt{\frac{SS}{N} - \frac{LS}{N}^2}.$$

- **Diameter**, defined as the largest distance between member points:

  $$R = \sqrt{\frac{2N \times SS - 2 \times LS^2}{N \times (N-1)}}.$$

When processing and summarizing continuously arriving stream data, the most recent observations are more important because they reflect the current state of the process generating the data. A popular approach in data stream clustering consists of defining a time window that covers the most recent data. The window models that have been used in the literature are the landmark model, sliding-window model, and damped model (Gama 2010).

Partitioning Clustering

K-means is the most widely used clustering algorithm. It constructs a partition of a set of objects into k clusters, that minimize some objective function, usually a squared error function, which imply round-shape clusters. The input parameter k is fixed and must be given in advance that limits its real applicability to streaming and evolving data.

Farnstrom et al. (2000) propose a *single-pass k-Means* algorithm. The main idea is to use a buffer where points of the dataset are kept in a compressed way. The data stream is processed in blocks. All available space on the buffer is filled with points from the stream. Using these points, find k-centers such that the sum of distances from data points to their closest center is minimized. Only the k-centroids (representing the clustering results) are retained, with the corresponding k-cluster features. Only the k-centroids (representing the clustering results) are retained, with the corresponding k-cluster features. In the next iterations, the buffer is initialized with the k-centroids, found in the previous iteration and the incoming data points from the stream. The *very fast k-means* algorithm (VFKM) (Domingos and Hulten 2001) uses the Hoeffding bound to determine the number of examples needed in each step of a k-means algorithm. VFKM runs as a sequence of k-means runs, with an increasing number of examples until the Hoeffding bound is satisfied.

Guha et al. (2003) present an analytical study on k-median clustering data streams. The proposed algorithm makes a single pass over the data stream and uses small space. It requires $O(nk)$ time and $O(n\epsilon)$ space where k is the number of centers, n is the number of points, and $\epsilon < 1$. They have proved that any k-median algorithm that achieves a constant factor approximation cannot achieve a better run time than $O(nk)$.

Micro-clustering

The idea of dividing the clustering process into two layers, where the first layer generates local models (micro-clusters) and the second layer generates global models from the local ones, is a powerful idea that has been used elsewhere.

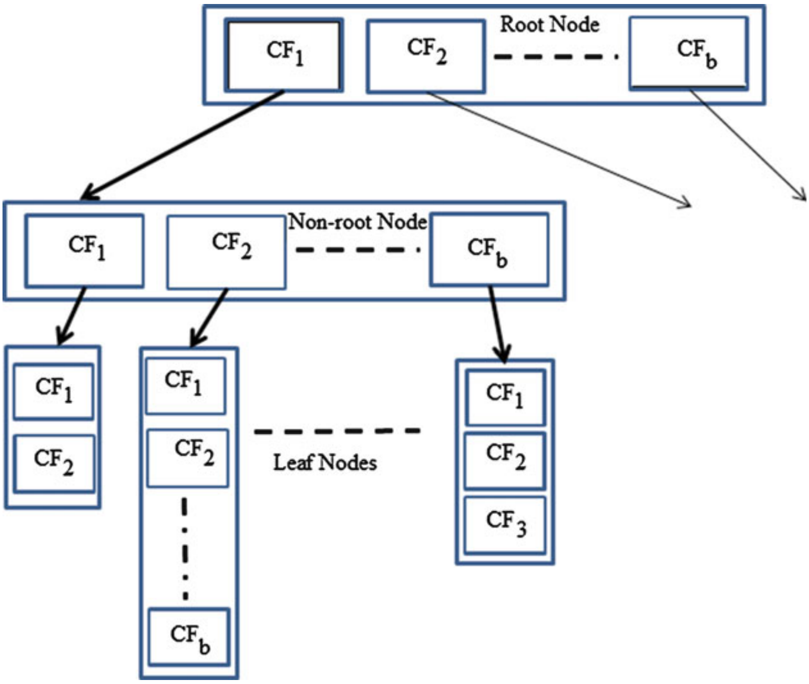
The BIRCH system (Zhang et al. 1996) builds a hierarchical structure of data, the CF-tree (see Fig. 2), where each node contains a set of cluster features. These CFs contain the sufficient statistics describing a set of points in the dataset and all information of the cluster features below in the tree. The system requires two user-defined parameters: b the branch factor or the maximum number of entries in each non-leaf node and T the maximum diameter (or radius) of any CF in a leaf node. The maximum diameter T defines the examples that can be *absorbed* by a CF. Increasing T, more examples can be absorbed by a micro-cluster and smaller CF-trees are generated.

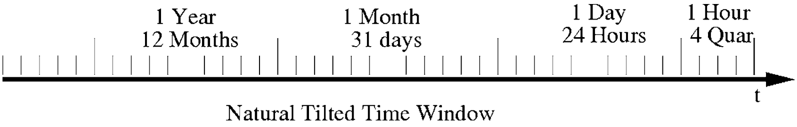
When an example is available, it traverses down the current tree from the root, till finding the appropriate leaf. At each non-leaf node, the example follows the *closest* CF path, with respect to norms L_1 or L_2. If the closest CF in the leaf cannot absorb the example, make a new CF entry. If there is no room for new leaf, split the parent node. A leaf node might be expanded due to the constrains imposed by B and T. The process consists of taking the two farthest CFs and creates two new leaf nodes. When traversing backup the CFs are updated.

Monitoring the Evolution of the Cluster Structure

The *CluStream* algorithm (Aggarwal et al. 2003) is an extension of the BIRCH system designed for data streams. Here, the CFs include temporal information: the time stamp of an example is treated as a feature. For each incoming data point, the distance to the centroids of existing CFs is computed. The data point is absorbed by an existing CF if the distance to the centroid falls

Clustering from Data Streams, Fig. 2 The clustering feature tree in BIRCH. **B** is the maximum number of CFs in a level of the tree





Clustering from Data Streams, Fig. 3 The figure presents a *natural tilted time window*. The most recent data is stored with high detail, while older data is stored in a compressed way. The degree of detail decreases with time

within the *maximum boundary* of the CF. The *maximum boundary* is defined as a factor t of the *radius* deviation of the CF; otherwise, the data point starts a new micro-cluster.

CluStream can generate approximate clusters for any user-defined time granularity. This is achieved by storing the CF at regular time intervals, referred to as snapshots. Suppose the user wants to find clusters in the stream based on a history of length h. The off-line component can analyze the snapshots stored at time t, the current time, and $(t - h)$ by using the addictive property of CF. An important problem is when to store the snapshots of the current set of micro-clusters. For example, the natural time frame stores snapshots each quarter, 4 quarters are aggregated in hours, 24 h are aggregated in days, etc. (Fig. 3). The aggregation level is domain dependent and explores the addictive property of CF. Along similar ideas, Kranen et al. (2011) present *ClusTree* that uses a weighted CF vector, which is kept into a hierarchical tree. *ClusTree* provides strategies for dealing with time constraints for anytime clustering, i.e., the possibility of interrupting the process of inserting new objects in the tree at any moment.

Tracking the Evolution of the Cluster Structure

Promising research lines are tracking change in clusters. Spiliopoulou et al. (2006) present system `MONIC`, for detecting and tracking change in clusters. MONIC assumes that a cluster is an object in a geometric space. It encompasses changes that involve more than one cluster, allowing for insights on cluster change in the whole clustering. The transition tracking mechanism is based on the degree of overlapping between the two clusters. The concept of *overlap* between two clusters, $\mathcal{X}$ and $\mathcal{Y}$, is defined as the normed number of common records weighted with the age of the records. Assume that cluster $\mathcal{X}$ was obtained at time t_1 and cluster $\mathcal{Y}$ at time t_2. The degree of overlapping between the two clusters is given by $overlap(X,Y) = \frac{\sum_{a \in X \cap Y} age(a,t_2)}{\sum_{x \in X} age(x,t_2)}$. The degree of overlapping allows inferring properties of the underlying data stream. Cluster transition at a given timepoint is a change in a cluster discovered at an earlier timepoint. MONIC considers *internal* and *external* transitions, which reflect the dynamics of the stream. Examples of cluster transitions include the cluster survives, the cluster is absorbed; a cluster disappears; and a new cluster emerges.

Recommended Reading

Ackermann MR, Martens M, Raupach C, Swierkot K, Lammersen C, Sohler C (2012) Streamkm++: a clustering algorithm for data streams. ACM J Exp Algorithmics 17:1

Aggarwal C, Han J, Wang J, Yu P (2003) A framework for clustering evolving data streams. In: Proceedings of twenty-ninth international conference on very large data bases. Morgan Kaufmann, St. Louis, pp 81–92

Domingos P, Hulten G (2001) A general method for scaling up machine learning algorithms and its application to clustering. In: Proceedings of international conference on machine learning. Morgan Kaufmann, San Francisco, pp 106–113

Farnstrom F, Lewis J, Elkan C (2000) Scalability for clustering algorithms revisited. SIGKDD Explor 2(1):51–57

Gama J (2010) Knowledge discovery from data streams. Chapman & Hall/CRC Press, Boca Raton

Gama J, Rodrigues PP, Lopes L (2011) Clustering distributed sensor data streams using local processing and reduced communication. Intell Data Anal 15(1):3–28

Guha S, Meyerson A, Mishra N, Motwani R, O'Callaghan L (2003) Clustering data streams: theory and practice. IEEE Trans Knowl Data Eng 15(3):515–528

Kranen P, Assent I, Baldauf C, Seidl T (2011) The clustree: indexing micro-clusters for anytime stream mining. Knowl Inf Syst 29(2):249–272

Silva JA, Faria E, Barros R, Hruschka E, Carvalho A, Gama J (2013) Data stream clustering: a survey. ACM Comput Surv 46(1):13

Spiliopoulou M, Ntoutsi I, Theodoridis Y, Schult R (2006) Monic: modeling and monitoring cluster transitions. In: Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining, Philadelphia, pp 706–711

Zhang T, Ramakrishnan R, Livny M (1996) Birch: an efficient data clustering method for very large databases. In: Proceedings of ACM SIGMOD international conference on management of data. ACM Press, New York, pp 103–114

Clustering of Nonnumerical Data

▶ Categorical Data Clustering

Clustering with Advice

▶ Correlation Clustering

Clustering with Constraints

▶ Correlation Clustering

Clustering with Qualitative Information

▶ Correlation Clustering

Clustering with Side Information

▶ Correlation Clustering

Coevolution

▶ Coevolutionary Learning

Coevolutionary Computation

▶ Coevolutionary Learning

Coevolutionary Learning

R. Paul Wiegand
University of Central Florida, Orlando, FL, USA

Synonyms

Coevolution; Coevolutionary computation

Definition

Coevolutionary learning is a form of evolutionary learning (see ▶ Evolutionary Algorithms) in which the fitness evaluation is based on interactions between individuals. Since the evaluation of an individual is dependent on interactions with other evolving entities, changes in the set of entities used for evaluation can affect an individual's ranking in a population. In this sense, coevolutionary fitness is *subjective*, while fitness in traditional evolutionary learning systems typically uses an *objective* performance measure.

Motivation and Background

Ideally, coevolutionary learning systems focus on relevant areas of a search space by making adaptive changes between interacting, concurrently evolving parts. This can be particularly helpful when problem spaces are very large – infinite search spaces in particular. Additionally, coevolution is useful when applied to problems when no intrinsic objective measure exists. The interactive nature of evaluation makes them natural methods to consider for problems such as the search for game-playing strategies (Fogel 2001). Finally, some coevolutionary systems appear natural for search spaces which contain certain kinds of complex structures (Potter 1997; Stanley 2004), since search on smaller components in a larger structure can be emphasized. In fact, there is reason to believe that coevolutionary systems may be well suited for uncovering complex structures within a problem (Bucci and Pollack 2002).

Still, the dynamics of coevolutionary learning can be quite complex, and a number of pathologies often plague naïve users. Indeed, because of the subjective nature of coevolution, it can be easy to apply a particular coevolutionary learning system without a clear understanding of what kind of solution one expects a coevolutionary algorithm to produce. Recent theoretical analysis suggests that a clear concept of solution and a careful implementation of an evaluation process consistent with this concept can produce a coevolutionary system capable of addressing many problems (de Jong and Pollack 2004; Ficici 2004; Panait 2006; Wiegand 2003). Accordingly, a great deal of research in this area focuses on evaluation and progress measurement.

Structure of Learning System

Coevolutionary learning systems work in much the same way that an evolutionary learning system works: individuals encode some aspect of potential solutions to a problem, those representatives are altered during search using genetic-like operators such as mutation and crossover, and the search is directed by selecting better individuals as determined by some kind of fitness assessment. These heuristic methods gradually refine solutions by repeatedly cycling through such steps, using the ideas of heredity and survival of the fittest to produce new generations of individuals, with increased quality of solution. Just as in traditional evolutionary computation, there are many choices available to the engineer in designing such systems. The reader is referred to the chapters relating to evolutionary learning for more details.

However, there are some fundamental differences between traditional evolution and coevolution. In coevolution, measuring fitness requires evaluating the interaction between multiple individuals. Interacting individuals may reside in the same population or in different populations; the interactive nature of coevolution evokes notions of cooperation and competition in entirely new ways; the choices regarding how to best conduct evaluation of these interactions for the purposes of selection are particularly important; and there are unique coevolutionary issues surrounding representation. In addition, because of its interactive nature, the dynamics of coevolution can lead to some well-known pathological behaviors, and particularly careful attention to implementation choices to avoid such conditions is generally necessary.

Multiple Versus Single Population Approaches

Coevolution can typically be broadly classified as to whether interacting individuals reside in different populations or in the same population.

In the case of multipopulation coevolution, measuring fitness requires evaluating how individuals in one population interact with individuals in another. For example, individuals in each population may represent potential strategies for particular players of a game, they may represent roles in a larger ecosystem (e.g., predators and prey), or they may represent components that are fitted into a composite assembly with other component then applied to a problem. Though individuals in different populations interact for the purposes of evaluation, they are typically otherwise independent of one another in the coevolutionary search process.

In single population coevolution, an individual in the population is evaluated based on his or her interaction with other individuals in the same population. Such individuals may again represent potential strategies in a game, but evaluation may require them to trade off roles as to which player they represent in that game. Here, individuals interact not only for evaluation, but also implicitly compete with one another as resources used in the coevolutionary search process itself.

There is some controversy in the field as to whether this latter type qualifies as "coevolution." Evolutionary biologists often define coevolution exclusively in terms of multiple populations; however, in biological systems, fitness is always subjective, while the vast majority of computational approaches to evolutionary learning involve objective fitness assessment – and this subjective/objective fitness distinction creates a useful classification.

To be sure, there are fundamental differences between how single population and multipopulation learning systems behave (Ficici 2004). Still, single population systems that employ subjective fitness assessment behave a lot more like multipopulation coevolutionary systems than like objective fitness based evolution. Moreover, historically, the field has used the term coevolution whenever fitness assessment is based on interactions between individuals, and a large amount of that research has involved systems with only one population.

Competition and Cooperation

The terms *cooperative* and *competitive* have been used to describe aspects of coevolution learning in at least three ways.

First and less commonly, these adjectives can describe qualitatively observed behaviors of potential solutions in coevolutionary systems, the results of some evolutionary process (e.g., "tit-for-tat" strategies, Axelrod 1984).

Second, problems are sometimes considered to be inherently competitive or cooperative. Indeed, game theory provides some guidance for making such distinctions. However, since in many kinds of problems little may be known about the actual structure of the payoff functions involved, we may not actually be able to classify the problem as definitively competitive or cooperative.

The final and by far most common use of the term is to distinguish algorithms themselves. Cooperative algorithms are those in which interacting individuals succeed or fail together, while competitive algorithms are those in which individuals succeed at the expense of other individuals.

Because of the ambiguity of the terms, some researchers advocate abandoning them altogether, instead focusing distinguishing terminology on the form a potential solution takes. For example, using the term ▸ compositional coevolution to describe an algorithm designed to return a solution composed of multiple individuals (e.g., a multiagent team) and using the term ▸ test-based coevolution to describe an algorithm designed to return an individual who performs well against an adaptive set of tests (e.g., sorting network). This latter pair of terms is a slightly different, though probably more useful distinction than the cooperative and competitive terms.

Still, it is instructive to survey the algorithms based on how they have been historically classified.

Examples of competitive coevolutionary learning include simultaneously learning sorting networks and challenging data sets in a predator–prey type relationship (Hillis 1991). Here, individuals in one population representing potential sorting networks are awarded a fitness score based on how well they sort opponent data sets from the other population. Individuals in the second population represent potential data sets whose fitness is based on how well they distinguish opponent sorting networks.

Competitive coevolution has also been applied to learning game-playing strategies (Fogel 2001; Rosin and Belew 1996). Additionally, competition has played a vital part in the attempts to coevolve complex agent behaviors (Sims 1994). Finally, competitive approaches have been applied to a variety of more traditional machine learning problems, for example, learning classifiers in one population and challenging subsets of exemplars in the other (Paredis 1994).

Potter developed a relatively general framework for cooperative coevolutionary learning, applying it first to static function optimization and later to neural network learning (Potter 1997). Here, each population contains individuals representing a portion of the network, and evolution of these components occurs almost independently, in tandem with one another, interacting only to be assembled into a complete network in order to obtain fitness. The decomposition of the network can be static and a priori, or dynamic in the sense that components may be added or removed during the learning process.

Moriarty et al. take a different, somewhat more adaptive approach to cooperative coevolution of neural networks (Moriarty and Miikkulainen 1997). In this case, one population represents potential network *plans*, while a second is used to acquire node information. Plans are evaluated based on how well they solve a problem with their collaborating nodes, and the nodes receive a share of this fitness. Thus, a node is rewarded for participating more with successful plans, and thus receives fitness only indirectly.

Evaluation

Choices surrounding how interacting individuals in coevolutionary systems are evaluated for the purposes of selection are perhaps the most important choices facing an engineer employing these methods. Designing the evaluation method involves a variety of practical choices, as well as a broader eye to the ultimate purpose of the algorithm itself.

Practical concerns in evaluation include determining the number of individuals with whom to interact, how those individuals will be chosen for the interaction, and how the selection will operate on the results of multiple interactions (Wiegand 2003). For example, one might determine the fitness of an individual by pairing him or her with all other individuals in the other populations (or the same population for single population approaches) and taking the average or maximum value of such evaluations as the fitness assessment. Alternatively, one may simply use the single best individual as determined by a previous generation of the algorithm, or a combination of those approaches. Random pairings between individuals is also common. This idea can be extended to use tournament evaluation where successful individuals from pairwise interactions are promoted and further paired, assigning fitness based on how far an individual progresses in the tournament. Many of these methods have been evaluated empirically on a variety of types of problems (Angeline and Pollack 1993; Bull 1997; Wiegand 2003).

However, the designing of the evaluation method also speaks to the broader issue of how to best implement the desired ▶ solution concept, (a criterion specifying which locations in the search space are solutions and which are not) (Ficici 2004). The key to successful application of coevolutionary learning is to first elicit a clear and precise solution concept and then design an algorithm (an evaluation method in particular) that implements such a concept explicitly.

A successful coevolutionary learner capable of achieving reliable progress toward a particular solution concept often makes use of an archive of individuals and an update rule for that archive that insists the distance to a particular solution concept decrease with every change to the archive. For example, if one is interested in finding game strategies that satisfy Nash equilibrium constraints, one might consider comparing new individuals to an archive of potential individual strategies found so far that together represent a potential Nash mixed strategy (Ficici 2004). Alternatively, if one is interested in maximizing the sum of an individual's outcomes over all tests, one may likewise employ an archive of discovered tests that candidate solutions are able to solve (de Jong 2004).

It is useful to note that many coevolutionary learning problems are multiobjective in nature. That is, ▸ underlying objectives may exist in such problems, each creating a different ranking for individuals depending on the set of tests being considered during evaluation (Bucci and Pollack 2002). The set of all possible underlying objectives (were it known) is sufficient to determine the outcomes on all possible tests. A careful understanding of this can yield approaches that create ideal and minimal evaluation sets for such problems (de Jong and Pollack 2004).

By acknowledging the link between multiobjective optimization and coevolutionary learning, a variety of evaluation and selection methods based on notions of multiobjective optimization have been employed. For example, there are selection methods that use Pareto dominance between candidate solutions and their tests as their basis of comparison (Ficici 2004). Additionally, such methods can be combined with archive-based approaches to ensure monotonicity of progress toward a Pareto dominance solution concept (de Jong and Pollack 2004).

Representation

Perhaps the core representational question in coevolution is the role that an individual plays. In test-based coevolution, an individual typically represents a potential solution to the problem or a test for a potential solution, whereas in compositional coevolution individuals typically represent a candidate component for a composite or ensemble solution.

Even in test-based approaches, the true solution to the problem may be expressed as a population of individuals, rather than a single individual. The population may represent a mixed strategy while individuals represent potential pure strategies for a game. Engineers using such approaches should be clear of the form of the final solution produced by the algorithm, and that this form is consistent with the prescribed solution concept.

In compositional approaches, the key issues tend to surround about how the problem is decomposed. In some algorithms, this decomposition is performed a priori, having different populations represent explicit components of the problem (Potter 1997). In other approaches, the decomposition is intended to be somewhat more dynamic (Moriarty and Miikkulainen 1997; Potter 1997). Still more recent approaches seek to harness the potential of compositional coevolutionary systems to search open-ended representational spaces by gradually *complexifying* the representational space during the search (Stanley 2004).

In addition, a variety of coevolutionary systems have successfully dealt with some inherent pathologies by representing populations in spatial topologies, and restricting selection and interaction using geometric constraints defined by those topologies (Pagie 1999). Typically, these systems involve overlaying multiple grids of individuals, applying selection within some neighborhood in a given grid, and evaluating interactions between individuals in different grids using a similar type of cross-population neighborhood. The benefits of these systems are in part due to their ability to naturally regulate loss of diversity and spread of interaction information by explicit control over the size and shape of these neighborhoods.

Pathologies and Remedies

Perhaps the most commonly cited pathology is the so-called *loss of gradient* problem, in which one population comes to severely dominate the others, thus creating a situation in which individuals cannot be distinguished from one another. The populations become disengaged and evolutionary progress may *stall* or *drift* (Watson and Pollack 2001). Disengagement most commonly occurs when distinguishing individuals are lost in the evolutionary process (*forgetting*), and the solution to this problem typically involves somehow retaining potentially informative, though possibly inferior quality individuals (e.g., archives).

Intransitivities in the reward system can cause some coevolutionary systems to exhibit *cycling* dynamics (Watson and Pollack 2001), where reciprocal changes force the system to orbit some part of a potential search space. The remedy to this problem often involves creating coevolutionary systems that change in response to traits in several other populations. Mechanisms introduced to produce such effects include *competitive fitness sharing* (Rosin and Belew 1996).

Another challenging problem occurs when individuals in a coevolutionary systems *overspecialize* on one underlying objective at the expense of other necessary objectives (Watson and Pollack 2001). In fact, overspecialization can be seen as a form of disengagement on some subset of underlying objectives, and likewise the repair to this problem often involves retaining individuals capable of making distinctions in as many underlying objectives as possible (Bucci and Pollack 2003).

Finally, certain kinds of compositional coevolutionary learning algorithms can be prone to *relative overgeneralization*, a pathology in which components that perform reasonably well in a variety of composite solutions are favored over those that are part of an optimal solution (Wiegand 2003). In this case, it is typically possible to bias the evaluation process toward optimal values by evaluating an individual in a variety of composite assemblies and assigning the best objective value found as the fitness (Panait 2006).

In addition to pathological behaviors in coevolution, the subjective nature of these learning systems creates difficulty in measuring progress. Since fitness is subjective, it is impossible to determine whether these relative measures indicate progress or stagnation when the measurement values do not change much. Without engaging some kind of external or objective measure, it is difficult to understand what the system is really doing. Obviously, if an objective measure exists then it can be employed directly to measure progress (Watson and Pollack 2001).

A variety of measurement methodologies have been employed when objective measurement is not possible. One method is to compare current individuals against all ancestral opponents (Cliff and Miller 1995). Another predator/prey based method holds *master tournaments* between all the best predators and all the best prey found during the search (Nolfi and Floreano 1998). A similar approach suggests maintaining the best individuals from each generation in each population in a *hall of fame* for comparison purposes (Rosin and Belew 1996). Still other approaches seek to record the points during the coevolutionary search in which a new dominant individual was found (Stanley 2004). A more recent approach advises looking at the *population differential*, examining all the information from ancestral generations rather than simply selecting a biased subset (Bader and Pollack 2005). Conversely, an alternative idea is to consider how well the *dynamics* of the best individuals in different populations reflect the fundamental *best response* curves defined by the problem (Popovici 2006).

With a clear solution concept, an appropriate evaluation mechanism implementing that concept, and practical progress measures in place, coevolution can be an effective and versatile machine learning tool.

Cross-References

▸ Evolutionary Algorithms

Recommended Reading

Angeline P, Pollack J (1993) Competitive environments evolve better solutions for complex tasks. In: Forest S (ed) Proceedings of the fifth international conference on genetic algorithms. Morgan Kaufmann, San Mateo, pp 264–270

Axelrod R (1984) The evolution of cooperation. Basic Books, New York

Bader-Natal A, Pollack J (2005) Towards metrics and visualizations sensitive to Coevolutionary failures. In: AAAI technical report FS-05-03 coevolutionary and coadaptive systems. AAAI Fall Symposium, Washington, DC

Bucci A, Pollack JB (2002) A mathematical framework for the study of coevolution. In: Poli R et al (eds) Foundations of genetic algorithms VII. Morgan Kaufmann, San Francisco, pp 221–235

Bucci A, Pollack JB (2003) Focusing versus intransitivity geometrical aspects of coevolution. In: Cantú-Paz E et al (eds) Proceedings of the 2003 genetic and evolutionary computation conference. Springer, Berlin, pp 250–261

Bull L (1997) Evolutionary computing in multi-agent environments: Partners. In: Bäck T (ed) Proceedings of the seventh international conference on genetic algorithms. Morgan Kaufmann, San Mateo, pp 370–377

Cliff D, Miller GF (1995) Tracking the red queen: measurements of adaptive progress in co-evolutionary simulations. In: Proceedings of the third European conference on artificial life. Springer, Berlin, pp 200–218

de Jong E (2004) The maxsolve algorithm for coevolution. In: Beyer H et al (eds) Proceedings of the 2005 genetic and evolutionary computation conference. ACM Press, New York, pp 483–489

de Jong E, Pollack J (2004) Ideal evaluation from coevolution. Evol Comput 12:159–192

Ficici SG (2004) Solution concepts in coevolutionary algorithms. PhD thesis, Brandeis University, Boston

Fogel D (2001) Blondie24: playing at the edge of artificial intelligence. Morgan Kaufmann, San Francisco

Hillis D (1991) Co-evolving parasites improve simulated evolution as an optimization procedure. Artificial life II, SFI studies in the sciences of complexity, vol 10. pp 313–324

Moriarty D, Miikkulainen R (1997) Forming neural networks through efficient and adaptive coevolution. Evol Comput 5:373–399

Nolfi S, Floreano D (1998) Co-evolving predator and prey robots: do “arm races” arise in artificial evolution? Artif Life 4:311–335

Pagie L (1999) Information integration in evolutionary processes. PhD thesis, Universiteit Utrecht, the Netherlands

Panait L (2006) The analysis and design of concurrent learning algorithms for cooperative multiagent systems. PhD thesis, George Mason University, Fairfax

Paredis J (1994) Steps towards co-evolutionary classification networks. In: Brooks RA, Maes P (eds) Artificial life IV, proceedings of the fourth international workshop on the synthesis and simulation of living systems. MIT Press, Cambridge, pp 359–365

Popovici E (2006) An analysis of multi-population coevolution. PhD thesis, George Mason University, Fairfax

Potter M (1997) The design and analysis of a computational model of cooperative co-evolution. PhD thesis, George Mason University, Fairfax

Rosin C, Belew R (1996) New methods for competitive coevolution. Evol Comput 5:1–29

Sims K (1994) Evolving 3D morphology and behavior by competition. In: Brooks RA, Maes P (eds) Artificial life IV, proceedings of the fourth international workshop on the synthesis and simulation of living systems. MIT Press, Cambridge, pp 28–39

Stanley K (2004) Efficient evolution of neural networks through complexification. PhD thesis, The University of Texas at Austin, Austin

Watson R, Pollack J (2001) Coevolutionary dynamics in a minimal substrate. In: Spector L et al (eds) Proceedings from the 2001 genetic and evolutionary computation conference. Morgan Kaufmann, San Francisco, pp 702–709

Wiegand RP (2003) An analysis of cooperative coevolutionary algorithms. PhD thesis, George Mason University, Fairfax

Collaborative Filtering

Collaborative Filtering (CF) refers to a class of techniques used in recommender systems, that recommend items to users that other users with similar tastes have liked in the past. CF methods are commonly sub-divided into *neighborhood-based* and *model-based* approaches. In neighborhood-based approaches, a subset of users are chosen based on their similarity to the active user, and a weighted combination of their ratings is used to produce predictions for this user. In contrast, model-based approaches assume an underlying structure to users' rating behavior, and induce predictive models based on the past ratings of all users.

Collection

► Class

Collective Classification

Galileo Namata, Prithviraj Sen, Mustafa Bilgic, and Lise Getoor
University of Maryland, College Park, MD, USA

Synonyms

Iterative classification; Link-based classification

Definition

Many real-world ▶classification problems can be best described as a set of objects interconnected via links to form a network structure. The links in the network denote relationships among the instances such that the class labels of the instances are often correlated. Thus, knowledge of the correct label for one instance improves our knowledge about the correct assignments to the other instances it connects to. The goal of collective classification is to *jointly* determine the correct label assignments of all the objects in the network.

Motivation and Background

Traditionally, a major focus of machine learning is to solve classification problems: given a corpus of documents, classify each according to its topic label; given a collection of e-mails, determine which are spam; given a sentence, determine the part-of-speech tag for each word; given a handwritten document, determine the characters, etc. However, much of the work in machine learning makes an *independent and identically distributed* (IID) assumption and focuses on predicting the class label of each instance in isolation. In many cases, however, the class labels whose values need to be determined can benefit if we know the correct assignments to related class labels. For example, it is easier to predict the topic of a webpage if we know the topics of the webpages that link to it, the chance of a particular word being a verb increases if we know that the previous word in the sentence is a noun, knowing the rest of the characters in a word can make it easier to identify an unknown character, etc. In the last decade, many researchers have proposed techniques that attempt to classify samples in a joint or collective manner instead of treating each sample in isolation and reported significant gains in classification accuracy.

Theory/Solution

Collective classification is a combinatorial optimization problem, in which we are given a set of nodes, $\mathcal{V} = \{v_1, \ldots, v_n\}$, and a neighborhood function $\mathcal{N}$, where $\mathcal{N}_i \subseteq \mathcal{V}\backslash\{v_i\}$, which describes the underlying network structure. Each node in $\mathcal{V}$ is a random variable that can take a value from an appropriate domain, $\mathcal{L} = \{l_1, \ldots, l_q\}$. $\mathcal{V}$ is further divided into two sets of nodes: $\mathcal{X}$, the nodes for which we know the correct values (observed variables), and $\mathcal{Y}$, the nodes whose values need to be determined. Our task is to label the nodes $y_i \in \mathcal{Y}$ with one of a small number of predefined labels in $\mathcal{L}$.

Even though it is only in the last decade that collective classification has entered the collective conscience of machine learning researchers, the general idea can be traced further back (Besag 1986). As a result, a number of approaches have been proposed. The various approaches to collective classification differ in the kinds of information they aim to exploit to arrive at the correct classification and their mathematical underpinnings. We discuss each in turn.

Relational Classification

Traditional classification concentrates on using the observed attributes of the instance to be classified. Relational classification (Slattery and Craven 1998) attempts to go a step further by classifying the instance using not only the instance's own attributes but also the instance's neighbors' attributes. For example, in a hypertext classification domain where we want to classify webpages, not only would we use the webpage's own words but we would also look at the webpages linking to this webpage using hyperlinks and their words to arrive at the correct class label. Results obtained using relational classification have been mixed. For example, even though there have been reports of classification accuracy gains using such techniques, in certain cases, these techniques can harm classification accuracy (Chakrabarti et al. 1998).

Iterative Collective Classification with Neighborhood Labels

A second approach to collective classification is to use the class labels assigned to the neighbor instead of using the neighbor's observed attributes. For example, going back to our hypertext classification example, instead of using the linking webpage's words, we would, in this case, use its assigned labels to classify the current webpage. Chakrabarti et al. (1998) illustrated the use of this approach and reported impressive classification accuracy gains. Neville and Jensen (2000) further developed the approach, and referred to the approach as iterative classification, and studied the conditions under which it improved classification performance (Jensen et al. 2004). Techniques for feature construction from the neighboring labels were developed and studied (Lu and Getoor 2003), along with methods that make use of *only* the label information (Macskassy and Provost 2007), as well as a variety of strategies for when to commit the class labels (McDowell et al. 2007).

Algorithm 1 depicts pseudo-code for a simple version of the iterative classification algorithm (ICA). The basic premise behind ICA is extremely simple. Consider a node $Y_i \in \mathcal{Y}$ whose value we need to determine and suppose we know the values of all the other nodes in its neighborhood $\mathcal{N}_i$ (note that $\mathcal{N}_i$ can contain both observed and unobserved variables). Then, ICA assumes that we are given a local classifier f that takes the values of $\mathcal{N}_i$ as arguments and returns a label value for Y_i from the class label set $\mathcal{L}$. For local classifiers f that do not return a class label but a goodness/likelihood value given a set of attribute values and a label, we simply choose the label that corresponds to the maximum goodness/likelihood value; in other words, we replace f with $\mathrm{argmax}_{l \in \mathcal{L}} f$. This makes the local classifier f extremely flexible, and we can use anything ranging from a decision tree to a ▸ support vector machine (SVM). Unfortunately, it is rare in practice that we know all values in $\mathcal{N}_i$, which is why we need to repeat the process iteratively, in each iteration, labeling each Y_i using the current best estimates of $\mathcal{N}_i$ and the local classifier f and continuing to do so until the assignments to the labels stabilize.

Most local classifiers are defined as functions whose argument consists of a fixed-length vector of attribute values. A common approach to circumvent such a situation is to use an aggregation operator such as count, mode, or prop, which measures the proportion of neighbors with a given label. In Algorithm 1, we use $\vec{a}_i$ to denote the vector encoding the values in $\mathcal{N}_i$ obtained after aggregation. Note that in the first ICA iteration, all labels y_i are undefined, and to initialize them we simply apply the local classifier to the observed attributes in the neighborhood of Y_i; this is referred to as "bootstrapping" in Algorithm 1.

Researchers in collective classification (Macskassy and Provost 2007; McDowell et al. 2007; Neville and Jensen 2000) have extended the simple algorithm described above and developed a version of Gibbs sampling that is easy to imple-

Algorithm 1 Iterative classification algorithm

Iterative Classification Algorithm (ICA)

```
for each node $Y_i \in \mathcal{Y}$ do {bootstrapping}
    {compute label using only observed nodes in $\mathcal{N}_i$}
    compute $\vec{a}_i$ using only $\mathcal{X} \cap \mathcal{N}_i$
    $y_i \leftarrow f(\vec{a}_i)$
end for
repeat {iterative classification}
    generate ordering $\mathcal{O}$ over nodes in $\mathcal{Y}$
    for each node $Y_i \in \mathcal{O}$ do
        {compute new estimate of $y_i$}
        compute $\vec{a}_i$ using current assignments to $\mathcal{N}_i$
        $y_i \leftarrow f(\vec{a}_i)$
    end for
until all class labels have stabilized or a threshold number of iterations have elapsed
```

ment and faster than traditional Gibbs sampling approaches. The basic idea behind this algorithm is to assume, just like in the case of ICA, that we have access to a local classifier f that can sample for the best label estimate for Y_i given all the values for the nodes in $\mathcal{N}_i$. We keep doing this repeatedly for a fixed number of iterations (a period known as "burn-in"). After that, not only do we sample for labels for each $Y_i \in \mathcal{Y}$ but we also maintain count statistics as to how many times we sampled label l for node Y_i. After collecting a predefined number of such samples, we output the best label assignment for node Y_i by choosing the label that was assigned the maximum number of times to Y_i while collecting samples.

One of the benefits of both variants of ICA is fairly simple to make use of any local classifier. Some of the classifiers used included the following: naïve Bayes (Chakrabarti et al. 1998; Neville and Jensen 2000), ▸ logistic regression (Lu and Getoor 2003), ▸ decision trees, (Jensen et al. 2004) and weighted-vote relational neighbor (Macskassy and Provost 2007). There is some evidence to indicate that discriminately trained local classifiers such as logistic regression tend to produce higher accuracies than others; this is consistent with results in other areas.

Other aspects of ICA that have been the subject of investigation include the ordering strategy to determine in which order to visit the nodes to relabel in each ICA iteration. There is some evidence to suggest that ICA is fairly robust to a number of simple ordering strategies such as random ordering, visiting nodes in ascending order of diversity of its neighborhood class labels, and labeling nodes in descending order of label confidences (Getoor 2005). However, there is also some evidence that certain modifications to the basic ICA procedure tend to produce improved classification accuracies. For example, both (Neville and Jensen 2000) and (McDowell et al. 2007) propose a strategy where only a subset of the unobserved variables are utilized as inputs for feature construction. More specifically, in each iteration, they choose the top-k most confident predicted labels and use only those unobserved variables in the following iteration's predictions, thus ignoring the less confident predicted labels. In each subsequent iteration, they increase the value of k so that in the last iteration, all nodes are used for prediction. McDowell et al. report that such a "cautious" approach leads to improved accuracies.

Collective Classification with Graphical Models

In addition to the approaches described above, which essentially focus on local representations and propagation methods, another approach to collective classification is by first representing the problem with a high-level global ▸ graphical model and then using learning and inference techniques for the graphical modeling approach to arrive at the correct classifications. These proposals include the use of both directed ▸ graphical models (Getoor et al. 2001) and undirected graphical models (Lafferty et al. 2001; Taskar et al. 2002). See ▸ statistical relational learning and Getoor and Taskar (2007) for a survey of various graphical models that are suitable for collective classification. In general, these techniques can use both neighborhood labels and observed attributes of neighbors. On the other hand, due to their generality, these techniques also tend to be less efficient than the iterative collective classification techniques.

One common way of defining such a global model uses a *pairwise Markov random field* (pairwise MRF) (Taskar et al. 2002). Let $G = (\mathcal{V}, E)$ denote a graph of random variables as before where $\mathcal{V}$ consists of two types of random variables, the unobserved variables, $\mathcal{Y}$, which need to be assigned domain values from label set $\mathcal{L}$, and observed variables $\mathcal{X}$ whose values we know (see ▸ Graphical Models). Let Ψ denote a set of *clique potentials*. Ψ contains three distinct types of functions:

- For each $Y_i \in \mathcal{Y}$, $\psi_i \in \Psi$ is a mapping $\psi_i : \mathcal{L} \rightarrow \Re_{\geq 0}$, where $\Re_{\geq 0}$ is the set of nonnegative real numbers.
- For each $(Y_i, X_j) \in E$, $\psi_{ij} \in \Psi$ is a mapping $\psi_{ij} : \mathcal{L} \rightarrow \Re_{\geq 0}$.

- For each $(Y_i, Y_j) \in E$, $\psi_{ij} \in \Psi$ is a mapping $\psi_{ij} : \mathcal{L} \times \mathcal{L} \to \mathfrak{R}_{\geq 0}$.

Let $\mathbf{x}$ denote the values assigned to all the observed variables in $\mathcal{V}$, and let x_i denote the value assigned to X_i. Similarly, let $\mathbf{y}$ denote any assignment to all the unobserved variables in $\mathcal{V}$, and let y_i denote a value assigned to Y_i. For brevity of notation, we will denote by ϕ_i the clique potential obtained by computing $\phi_i(y_i) = \psi_i(y_i) \prod_{(Y_i, X_j) \in E} \psi_{ij}(y_i)$. We are now in a position to define a pairwise MRF.

Definition 1 A *pairwise Markov random field* (MRF) is given by a pair $\langle G, \Psi \rangle$ where G is a graph and Ψ is a set of clique potentials with ϕ_i and ψ_{ij} as defined above. Given an assignment $\mathbf{y}$ to all the unobserved variables $\mathcal{Y}$, the pairwise MRF is associated with the probability distribution $P(\mathbf{y}|\mathbf{x}) = \frac{1}{\mathcal{Z}(\mathbf{x})} \prod_{Y_i \in \mathcal{Y}} \phi_i(y_i) \prod_{(Y_i, Y_j) \in E} \psi_{ij}(y_i, y_j)$ where $\mathbf{x}$ denotes the observed values of $\mathcal{X}$ and $\mathcal{Z}(\mathbf{x}) = \sum_{\mathbf{y}'} \prod_{Y_i \in \mathcal{Y}} \phi_i\left(y'_i\right) \prod_{(Y_i, Y_j) \in E} \psi_{ij}\left(y'_i, y'_j\right)$.

Given a pairwise MRF, it is conceptually simple to extract the best assignments to each unobserved variable in the network. For example, we may adopt the criterion that the best label value for Y_i is simply the one corresponding to the highest marginal probability obtained by summing over all other variables from the probability distribution associated with the pairwise MRF. Computationally, however, this is difficult to achieve since computing one marginal probability requires summing over an exponentially large number of terms, which is why we need approximate inference algorithms. Hence, approximate inference algorithms are typically employed, the two most common being *loopy belief propagation* (LBP) and *mean-field relaxation labeling*.

Applications

Due to its general applicability, collective classification has been applied to a number of real-world problems. Foremost in this list is document classification. Chakrabarti et al. (1998) was one of the first to apply collective classification to corpora of patents linked via hyperlinks and reported that considering attributes of neighboring documents actually hurts classification performance. Slattery and Craven (1998) also considered the problem of document classification by constructing features from neighboring documents using an ▶ inductive logic programming rule learner. Yang et al. (2002) conducted an in-depth investigation over multiple datasets commonly used for document classification experiments and identified different patterns. Other applications of collective classification include object labeling in images (Hummel and Zucker 1983), analysis of spatial statistics (Besag 1986), iterative decoding (Berrou et al. 1993), part-of-speech tagging (Lafferty et al. 2001), classification of hypertext documents using hyperlinks (Taskar et al. 2002), link prediction (Getoor et al. 2002; Taskar et al. 2003b), optical character recognition (Taskar et al. 2003a), entity resolution in sensor networks (Chen et al. 2003), predicting disulfide bonds in protein molecules (Taskar et al. 2005), segmentation of 3D scan data (Anguelov et al. 2005), and classification of e-mail speech acts (Carvalho and Cohen 2005). Recently, there have also been attempts to extend collective classification techniques to the semi-supervised learning scenario (Lu and Getoor 2003b; Macskassy 2007; Xu et al. 2006).

Cross-References

▶ Decision Tree
▶ Inductive Logic Programming
▶ Learning From Structured Data
▶ Relational Learning
▶ Semi-supervised Learning
▶ Statistical Relational Learning

Recommended Reading

Anguelov D, Taskar B, Chatalbashev V, Koller D, Gupta D, Heitz G et al (2005) Discriminative learning of Markov random fields for segmentation of 3D scan data. In: IEEE computer society conference on computer vision and pattern recognition, San Diego. IEEE Computer Society, Washington, DC

Berrou C, Glavieux A, Thitimajshima P (1993) Near Shannon limit error-correcting coding and decoding: Turbo codes. In: Proceedings of IEEE international communications conference, Geneva. IEEE

Besag J (1986) On the statistical analysis of dirty pictures. J R Stat Soc B-48:259–302

Carvalho V, Cohen WW (2005) On the collective classification of email speech acts. In: Special interest group on information retrieval, Salvador. ACM

Chakrabarti S, Dom B, Indyk P (1998) Enhanced hypertext categorization using hyperlinks. In: International conference on management of data, Seattle. ACM, New York

Chen L, Wainwright M, Cetin M, Willsky A (2003) Multitarget multisensor data association using the tree-reweighted max-product algorithm. In: SPIE Aerosense conference, Orlando

Getoor L (2005) Link-based classification. In: Advanced methods for knowledge discovery from complex data. Springer, New York

Getoor L, Taskar B (eds) (2007) Introduction to statistical relational learning. MIT, Cambridge

Getoor L, Segal E, Taskar B, Koller D (2001) Probabilistic models of text and link structure for hypertext classification. In: Proceedings of the IJCAI workshop on text learning: beyond supervision, Seattle

Getoor L, Friedman N, Koller D, Taskar B (2002) Learning probabilistic models of link structure. J Mach Learn Res 3:679–707

Hummel R, Zucker S (1983) On the foundations of relaxation labeling processes. IEEE Trans Pattern Anal Mach Intell 5:267–287

Jensen D, Neville J, Gallagher B (2004) Why collective inference improves relational classification. In: Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining, Seattle. ACM

Lafferty JD, McCallum A, Pereira FCN (2001) conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proceedings of the international conference on machine learning, Washington, DC. Morgan Kaufmann, San Francisco

Lu Q, Getoor L (2003a) Link based classification. In: Proceedings of the international conference on machine learning, Washington, DC. AAAI

Lu Q, Getoor L (2003b) Link-based classification using labeled and unlabeled data. In: ICML workshop on the continuum from labeled to unlabeled data in machine learning and data mining, Washington, DC

Macskassy S, Provost F (2007) Classification in networked data: a toolkit and a univariate case study. J Mach Learn Res 8:935–983

Macskassy SA (2007) Improving learning in networked data by combining explicit and mined links. In: Proceedings of the twenty-second AAAI conference on artificial intelligence, Vancouver. AAAI

McDowell LK, Gupta KM, Aha DW (2007) Cautious inference in collective classification. In: Proceedings of the twenty-second AAAI conference on artificial intelligence, Vancouver. AAAI

Neville J, Jensen D (2007) Relational dependency networks. J Mach Learn Res 8:653–692

Neville J, Jensen D (2000) Iterative classification in relation data. In: Workshop on statistical relational learning. AAAI

Slattery S, Craven M (1998) Combining statistical and relational methods for learning in hypertext domains. In: International conferences on inductive logic programming, Madison. Springer, London

Taskar B, Abbeel P, Koller D (2002) Discriminative probabilistic models for relational data. In: Proceedings of the annual conference on uncertainty in artificial intelligence, Edmonton. Morgan Kauffman, San Francisco

Taskar B, Guestrin C, Koller D (2003a) Max-margin Markov networks. In: Neural information processing systems. MIT, Cambridge

Taskar B, Wong MF, Abbeel P, Koller D (2003b) Link prediction in relational data. In: Natural information processing systems. MIT, Cambridge

Taskar B, Chatalbashev V, Koller D, Guestrin C (2005) Learning structured prediction models: a large margin approach. In: Proceedings of the international conference on machine learning, Bonn. ACM, New York

Xu L, Wilkinson D, Southey F, Schuurmans D (2006) Discriminative unsupervised learning of structured predictors. In: Proceedings of the international conference on machine learning, Pittsburgh. ACM, New York

Yang Y, Slattery S, Ghani R (2002) A study of approaches to hypertext categorization. J Intell Inf Syst 18(2–3):219–241

Commercial Email Filtering

▶ Text Mining for Spam Filtering

Committee Machines

▶ Ensemble Learning

Community Detection

▶ Group Detection

Comparable Corpus

A comparable corpus (pl. corpora) is a document collection composed of two or more disjoint subsets, each written in a different language, such that documents in each subset are on a same topic as the documents in the others. The prototypical example of a comparable corpora is a collection of newspaper article written in different languages and reporting about the same events: while they will not be, strictly speaking, the translation of one another, they will share most of the semantic content. Some methods for cross-language text mining rely, totally or partially, on the statistical properties of comparable corpora.

Comparison Training

▸ Preference Learning

Competitive Coevolution

▸ Test-Based Coevolution

Competitive Learning

A *Competitive learning* is an ▸ artificial neural network learning process where different neurons or processing elements compete on who is allowed to learn to represent the current input. In its purest form competitive learning is in the so-called winner-take-all networks where only the neuron that best represents the input is allowed to learn. Since all neurons learn to better represent the kinds of inputs they already are good at representing, they become specialized to represent different kinds of inputs. For vector-valued inputs and representations, the input becomes quantized to the unit having the closest representation (model), and the representations are adapted to minimize the representation error using stochastic gradient descent.

Competitive learning networks have been studied as models of how receptive fields and feature detectors, such as orientation-selective visual neurons, develop in neural networks. The same process is at work in online ▸ K-means clustering, and variants of it in ▸ Self-Organizing Maps (SOM) and the EM algorithm of mixture models.

Complex Adaptive System

▸ Complexity in Adaptive Systems

Complexity in Adaptive Systems

Jun He
Aberystwyth University, Aberystwyth, UK

Abstract

The complexity in adaptive systems is classified into two types: internal complexity for model complexity and external complexity for data complexity. As an application, the two concepts are put into the background of learning and are used to explain statistical learning.

Synonyms

Adaptive system; Complex adaptive system

Definition

An adaptive system, or complex adaptive system, is a special case of complex systems, which is able to adapt its behavior according to changes in its environment or in parts of the system itself. In this way, the system can improve its performance through a continuing interaction with its environment. The concept of complexity in an adaptive system is used to analyze the interactive relationship between the system and its environment, which can be classified into two types: internal complexity for model complexity and external complexity for data complexity. The internal complexity is defined by the amount of input, information, or energy that the system receives from its environment. The external complexity refers to the complexity of how the system represents these inputs through its internal process.

Motivation and Background

Adaptive systems range from natural systems to artificial systems (Holland 1992, 1995; Mitchell 1992). Examples of natural systems include ant colonies, ecosystem, the brain, neural network and immune system, cell, and developing embryo; examples of artificial systems include the stock market, social system, manufacturing businesses, and human social group-based endeavor in a cultural and social system such as political parties or communities. All these systems have a common feature: they can adapt to their environment.

An adaptive system is adaptive in that way it has the capacity to change its internal structure for adapting the environment. It is complex in the sense that it interacts with their environment. The interaction between an adaptive system and its environment is dynamic and nonlinear. Complexity emerges from the interaction among the system and environment and the elements of the system, where the emergent macroscopic patterns are more complex than the sum of the these low-level (microscopic) elements encompassed in the system. Understanding the evolution and development of adaptive systems still faces many mathematical challenges (Levin 2003).

The concepts of external and internal complexity are used to analyze the relation between an adaptive system and its environment. The description given below is based on Jürgen Jost's work (Jost 2004), which introduced these two concepts and applied the theoretical framework to the construction of learning models, e.g., to design neural network architectures. In the following, the concepts are mainly applied to analyze the interaction between the system and its environment. The interaction among individual elements of the system is less discussed; however, the concepts can be explored in that situation too.

Theory

Adaptive System, Environment, and Regularities

The environment of an adaptive system is more complex than the system itself and its changes cannot be completely predictable for the system. However, the changes of the environment are not purely random and noisy; there exist regularities in the environment. An adaptive system can recognize these regularities; and depending on these regularities, the system will express them through its internal process in order to adapt to the environment.

The input that an adaptive system receives or extracts from its environment usually includes two parts: one is the part with regularities and another is that appears random to the system. The part of regularities is useful and meaningful. An adaptive system will represent these regularities by internal processes. But the part of random input is useless, and even at the worst it will be detrimental for an adaptive system. However, it will depend on the adaptive system's internal model of the external environment for how to determine which part of input is meaningful and regular and which part is random and devoid of meaning and structure.

An adaptive system will translate the external regularities into its internal ones, and only the regularities are useful to the system. The system tries to extract regularities as many as possible and to represent these regularities as efficiently as possible in order to make optimal use of its capacity.

The notions of external complexity and internal complexity are used to investigate these two complementary aspects conceptually and quantitatively. In terms of these notions, an adaptive system aims to increase their external complexity and reduce their internal complexity.

The two processes operate on their own time scale but are intricately linked and mutually dependent on each other. For example, the internal complexity will be only reduced if the external complexity is fixed. Under fixed inputs received from the external environment, an adaptive system can represent these inputs systems more efficiently and optimize its internal structure. If the external complexity is increased, e.g., additional new input is required to handle by the system, then it is necessary to increase its internal complexity.

The increase of internal complexity may occur through the creation of redundancy in the existing adaptive system, e.g., to duplicate some internal structures and then enable the system to handle more external input. Once the input is fixed, the adaptive then will represent the input as efficiently as possible and reduce the internal input. The decrease of internal complexity can be achieved through discarding some input as meaningless and irrelevant, e.g., leaving some regularities out, for the purpose.

Since the inputs relevant for the systems are those which can be reflected in the internal model, the external complexity is not equivalent to the amount of raw data received from the environment. In fact, it is only relevant to the inputs which can be processed in the internal model or observations in some adaptive systems. Thus the external complexity ultimately is decided by the internal model constructed by the system.

External and Internal Complexity

External complexity means data complexity, which is used to measure the amount of input received from the environment for the system to handle and process. Such a complexity can be measured by entropy in the term of information theory.

Internal complexity is model complexity, which is used to measure the complexity of a model for representing the input or information received by the system.

The aim of the adaptive system is to obtain an efficient model as simple as possible, with the capacity to handle as much input as possible. On the one hand, the adaptive system will try to maximize its external complexity and then to adapt to its environment in a maximal way and, on the other hand, to minimize its internal complexity and then to construct a model to process the input in the most efficient way.

These two aims sometimes seem conflicting, but such a conflict can be avoided when these two processes operate on different time scales. If given a model, the system will organize the input data and try to increase its ability to deal input from its environment and then increase its external complexity. If given the input, conversely, it tries to simplify its model which represents that input and thus to decrease the internal complexity. The meaning of the input is relevant to the time scale under investigation. On a short time scale, for example, the input may consist of individual signals, but on a long time scale, it will be a sequence of signals, which satisfies a probability distribution. A good internal model tries to express regularities in the input sequence, rather than several individual signals. And the decrease of internal complexity will happen on this time scale.

A formal definition of the internal and external complexity concepts is based on the concept of entropy from statistical mechanics and information theory. Given a model θ, the system can model data as with $X(\theta) = (X_1, \cdots, X_k)$, which is assumed to have an internal probability distribution $P(X(\theta))$ so that entropy can be computed. The external complexity is defined by

$$-\sum_{i=1}^{k} P(X_i(\theta)) \log_2 P(X_i(\theta)). \qquad (1)$$

An adaptive system tries to maximize the above external complexity.

The probability distribution $P(X(\theta))$ is for quantifying the information value of the data $X(\theta)$. The value of information can be described in other approaches, e.g., the length of the representation of the data in the internal code of the system (Rissanen 1989/1998). In this case, the optimal coding is a consequence of the minimization of internal complexity, and then the length of the representation of data $X_i(\theta)$ behaves like $\log_2 P(X(\theta))$ (Rissanen 1989/1998).

On a short time scale, for a given model θ, the system tries to increase the amount of meaningful input information $X(\theta)$. On a long time scale, when the input is given, e.g., when the system has gathered a set of input on a time scale with a stationary probability distribution of input patterns Ξ, then the model should be improved to handle the input as efficiently as possible and reduce the complexity of model. This complexity, or internal complexity, is defined by

$$-\sum_{i=1}^{k} P(\Xi_i \mid \theta) \log_2 P(\Xi_i \mid \theta) - \log_2 P(\theta), \quad (2)$$

with respect to the model θ.

If Rissanen's minimum description length principle (Rissanen 1989/1998) is applied to the above formula, then the optimal model will satisfy the following variation problem:

$$\min_{\theta} (-\log_2 P(\Xi \mid \theta) - \log_2 P(\theta)) . \quad (3)$$

Here in the above minimization problem, there are two objectives to minimize. The first term is to measure how efficiently the model represents or encodes the data and the second one to how complicated the model is. In computer science, this latter term corresponds to the length of the program required to encode the model.

The concepts of external and internal complexity can be applied into a system divided into subsystems. In this case, some internal part of the original whole system will become external to a subsystem. Thus the internal input of a subsystem consists of original external input and also input from the rest of the system, i.e., other subsystems.

Application: Learning

The discussion of these two concepts, external and internal complexity, can be put into the background of learning. In statistical learning theory (Vapnik 1998), the criterion for evaluating a learning process is the expected prediction error of future data by the model based on training data set with partial and incomplete information. The task is to construct a probability distribution drawn from an a priori specific class for representing the distribution underlying the input data received. Usually if a higher error is produced by a model on the training data, then a higher error will be expected on the future data. The error will depend on two factors: one is the accuracy of the model on the training data set and another is the simplicity of the model itself. The description of the data set can be split into two parts, the regular part, which is useful in construct the model, and the random part, which is a noise to the model.

The learning process fits very well into the theory framework of internal and external complexity. If the model is too complicated, it will bring the risk of over-fitting the training dada. In this case, some spurious or putative regularity is incorporated into the model, which will not appear in the future data. The model should be constrained within some model class with bounded complexity. This complexity in this context of statistical learning theory is measured by the Vapnik-Chervonenkis dimension (Vapnik 1998). Under the simplest form of statistical learning theory, the system aims at finding a representation with smallest error in a class with given complexity constraints; and then the model should minimize the expected error on future data and also over-fitting error.

The two concepts of over-fitting and leaving out regularities can be distinguished in the following sense. The former is caused by the noise in the data, i.e., the random part of the data, and this leads to putative regularities, which will not appear in the future data. The latter, leaving out regularities, means the system can forgo some part of regularities in the data, or it is possible to make data compression. Thus leaving out regularities can be used to simplify the model and reduce the internal complexity. However, a problem is still waiting for answer here, that is, what regularities in the data set are useful for data compression and also meaningful for future prediction and what parts are random to the model.

The internal complexity is the model complexity. If the internal complexity is chosen too small, then the model does not have enough capacity to represent all the important features of the data set. If the internal complexity is too large, on the other hand, then the model doesn't represent the data efficiently. The internal complexity is preferably minimized under appropriate constraints on the adequacy of the representation of data. This is consistent with Rissanen's principle of minimum description length (Rissanen 1989/1998), to represent the given data set in the most efficient way. Thus a good model is both to simplify the model itself and to represent the data efficiently.

The external complexity is the data complexity which should be large to represent the input accurately. This is related to Jaynes' principle of maximizing the ignorance (Jaynes 1957), where a model for representing data should have the maximal possible entropy under the constraint that all regularities can be reproduced. In this way, putative regularities could be eliminated in the model. However, this principle should be applied with some conditions; as argued by Gell-Mann and Lloyd (1996), it cannot eliminate the essential regularities in the data, and an overly complex model should be avoided.

For some learning system, only a selection of data is gathered and observed by the system. Thus a middle term, observation, is added between model and data. The concept of observation refers to the extraction of value of some specific quantity from a given data or data pool. What a system can observe depends on its internal structure and its general model of the environment. The system doesn't have direct access to the raw data, but through constructing a model of the environment solely on the basis of the values of its observation.

For such kind of learning system, Jaynes' principle (Jaynes 1957) is still applicable for increasing the external complexity. For the given observation made on a data set, the maximum entropy representation should be selected. However, this principle is still subject to the modification of Gell-Mann and Lloyd (1996) to the principle, where the model should not lose the essential regularities observed in the data.

In contrast, the observations should be selected to reduce the internal complexity. Given a model, if the observation can be made on a given data set, then these observations should be selected so as to minimize the resulting entropy of the model, with the purpose of minimizing the uncertainty left about the data. Thus it leads to reduce complexity.

In most cases, the environment is dynamic, i.e., the data set itself can be varied, then the external complexity should be maximized again. Thus the observation should be chosen for maximal information gain extracted from the data to increase the external complexity. Jaynes' principle (Jaynes 1957) can be applied as the same as in previous discussion. But in a longer time scale, when the input reaches some stationary distribution, the model should be simplified to reduce its internal complexity.

Recommended Reading

Gell-Mann, M and Lloyd, S (1996) Information measures, effective complexity, and total information. Complexity 2(1):44–52

Holland J (1992) Adaptation in natural and artificial systems. MIT Press, Cambridge

Holland J (1995) Hidden order: how adaptation builds complexity. Addison-Wesley, Redwood City

Jaynes, E. (1957) Information theory and statistical mechanics. Physical Review 106(4):620–630

Jost J (2004) External and internal complexity of complex adaptive systems. Theory Biosci 123(1):69–88

Levin S (2003) Complex adaptive systems: exploring the known, the unknown and the unknowable. Bull Am Math Soc 40(1):3–19

Rissanen J (1989/1998) Stochastic complexity in statistical inquiry. World Scientific, Singapore

Vapnik V (1998) Statistical learning theory. Wiley, New York (1998)

Mitchell W. M (1992) Complexity: the emerging science at the edge of order and chaos. Simon and Schuster, New York

Complexity of Inductive Inference

Sanjay Jain[1] and Frank Stephan[2]
[1]School of Computing, National University of Singapore, Singapore, Singapore
[2]Department of Mathematics, National University of Singapore, Singapore, Singapore

Definition

In ▸ inductive inference, the complexity of learning can be measured in various ways: by the number of hypotheses issued in the worst case until the correct hypothesis is found, by the number of data items to be consumed or to be memorized in order to learn in the worst case, by the Turing degree of oracles needed to learn the class under a certain criterion, and by the intrinsic complexity which is – like the Turing degrees in recursion theory – a way to measure the complexity of classes by using reducibilities between them.

Detail

We refer the reader to the article ▸ Inductive Inference for basic definitions in inductive inference and the notations used below. Let $\mathbb{N}$ denote the set of natural numbers. Let $\varphi_0, \varphi_1, \ldots$ denote a fixed acceptable numbering of the partial-recursive functions (Rogers 1967). Let $W_i = \text{domain}(\varphi_i)$.

Mind Changes and Anomalies

The first measure of complexity of learning can be considered as the number of mind changes needed before the learner converges to its final hypothesis in the **TxtEx** model of learning. The number of mind changes by a learner M on a text T can be counted as card $(\{m : ? \neq \text{M}(T[m]) \neq \text{M}(T[m+1])\})$. A learner M $\mathbf{TxtEx}_n$ learns a class $\mathcal{L}$ of languages if M **TxtEx** learns $\mathcal{L}$ and for all $L \in \mathcal{L}$, for all texts T for L, M makes at most n mind changes on T. $\mathbf{TxtEx}_n$ is defined as the collection of language classes which can be $\mathbf{TxtEx}_n$ identified (see Case and Smith (1983) for details).

Consider the class of languages $\mathcal{L}_n = \{L : \text{card}(L) \leq n\}$. It can be shown that $\mathcal{L}_{n+1} \in \mathbf{TxtEx}_{n+1} - \mathbf{TxtEx}_n$.

Now consider anomalous learning. A class $\mathcal{C}$ is $\mathbf{TxtEx}_b^a$ learnable if there is a learner, which makes at most b mind changes (where $b = *$ denotes that the number of mind changes is finite on each text for a language in the class, but not necessarily bounded by a constant) and whose final hypothesis is allowed to make up to a errors (where $a = *$ denotes finitely many errors). For these learning criteria, we get a two-dimensional hierarchy on what can be learnt. Let $\mathcal{C}_n = \{f : \varphi_{f(0)} =^n f\}$. For a total function f, let $L_f = \{\langle x, f(x) \rangle : x \in \mathbb{N}\}$, where $\langle \cdot, \cdot \rangle$ denotes a computable pairing function: a bijective mapping from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$. Let $\mathcal{L}_{\mathcal{C}} = \{L_f : f \in \mathcal{C}\}$. Then, one can show that $\mathcal{L}_{\mathcal{C}_{n+1}} \in \mathbf{TxtEx}_0^{n+1} - \mathbf{TxtEx}^n$. Similarly, if we consider the class $\mathcal{S}_n = \{f : \text{card}(\{m : f(m) \neq f(m+1)\}) \leq n\}$, then one can show that $\mathcal{L}_{\mathcal{S}_{n+1}} \in \mathbf{TxtEx}_{n+1}^0 - \mathbf{TxtEx}_n^*$ (we refer the reader to Case and Smith (1983) for a proof of the above).

Data and Time Complexity

Wiehagen (1986) considered the complexity of the number of data needed for learning. Regarding time complexity, one should note the result by Pitt (1989) that any **TxtEx**-learnable class of languages can be **TxtEx**-learnt by a learner that has time complexity (with respect to the size of the input) bounded by a linear function. This result is achieved by a delaying trick, where the learner just repeats its old hypothesis unless it has enough time to compute its later hypothesis. This seriously effects what one can say about time complexity of learning. One proposal made by Daley and Smith (1986) is to consider the total time used by the learner until its sequence of hypotheses converges, resulting in a possibly more reasonable measure of time in the complexity of learning.

Iterative and Memory-Bounded Learning

Another measure of complexity of learning can be considered when one restricts how much past data a learner can remember. Wiehagen (1976) introduced the concept of *iterative* learning in which the learner cannot remember any past data. Its new hypothesis is based only on its previous conjecture and the new datum it receives. In other words, there exists a recursive function F such that $\text{M}(T[n+1]) = F(\text{M}(T[n]), T(n))$, for all texts T and for all n. Here, $\text{M}(T[0])$ is some fixed value, say the symbol "?" which is used by the learner to denote the absence of a reasonable conjecture. It can be shown that being iterative restricts the learning capacity of learners. For example, let $L_e = \{2x : x \in \mathbb{N}\}$ and let $\mathcal{L} = \{L_e\} \cup \{\{S \cup \{2n+1\}\} : n \in \mathbb{N}, S \subseteq L_e$, and $\max(S) \leq n\}$; then $\mathcal{L}$ can be shown to be **TxtEx** learnable but not iteratively learnable.

Memory-bounded learning (see Lange and Zeugmann 1996) is an extension of memory-limited learning, where the learner is allowed to memorize up to some fixed number of elements seen in the past. Thus, M is an m-memory-bounded learner if there exists a function mem and two computable functions mF and F such that, for all texts T and all n:

- $mem(T[0]) = \emptyset$;
- $\mathrm{M}(T[n+1]) = F(\mathrm{M}(T[n]), mem(T[n]), T(n+1))$;
- $mem(T[n+1]) = mF(\mathrm{M}(T[n]), mem(T[n]), T(n+1))$;
- $mem(T[n+1]) - mem(T[n]) \subseteq \{T(n+1)\}$;
- $\mathrm{card}(mem(T[n])) \leq m$.

It can be shown that the criteria of inference based on **TxtEx** learning by m-memory-bounded learners form a proper hierarchy.

Besides memorizing some past elements seen, another way to address this issue is by giving feedback to the learner (see Case et al. 1999) on whether some element has appeared in the past data. A feedback learner is an iterative learner, which is additionally allowed to query whether certain elements appeared in earlier data. An n-feedback learner is allowed to make n such queries at each stage (when it receives the new input datum). Thus, M is an m-feedback learner if there exist computable functions Q and a F such that, for all texts T and all n:

- $Q(\mathrm{M}(T[n]), T(n))$ is defined and is a set of m elements
- If $Q(\mathrm{M}(T[n]), T(n)) = (x_1, x_2, \ldots, x_m)$, then $\mathrm{M}(T[n+1]) = F(\mathrm{M}(T[n]), T(n), y_1, y_2, \ldots, y_m)$, where $y_i = 1$ iff $x_i \in \mathrm{ctnt}(T[n])$.

Again, it can be shown that allowing more feedback gives greater learning power, and thus one can get a hierarchy based on the amount of feedback allowed.

Complexity of Final Hypothesis

Another possibility on complexity of learning is to consider the complexity or size of the final grammar output by the learner. Freivalds (1975) considered the case when the final program/grammar output by the learner is minimal: that is, there is no smaller index that accepts/generates the same language. He showed that this severely restricts the learning capacity of learners. Not only that, the learning capacity depends on the acceptable programming system chosen, unlike the case for most other criteria of learning such as **TxtEx** or **TxtBc**, which are independent of the acceptable programming system chosen. In particular, there are acceptable programming systems in which only classes containing finitely many infinite languages can be learnt using minimal final grammars (see Freivalds 1975; Jain and Sharma 1993). Chen (1982) considered a modification of such a paradigm where one considers convergence to nearly minimal grammars rather than minimal. That is, instead of requiring that the final grammars are minimal, one requires that they are within a recursive function h of minimal. Here h may depend on the class being learnt. Chen showed that this allows one to have the criteria of minimal learnability to be independent of the acceptable programming system chosen. However, one can show that some simple classes are not minimally learnable. An example of such a class is the class $\mathcal{L}_{\mathcal{C}}$ which is derived from $\mathcal{C} = \{f : \forall^\infty x[f(x) = 0]\}$, the class of all functions which are almost everywhere 0.

Intrinsic Complexity

Another way to consider complexity of learning is to consider relative complexity in a way similar to how one considers Turing reductions in computability theory. Such a notion is called intrinsic complexity of the class. This was first considered by Freivalds et al. (1995) for function learning. Jain and Sharma (1996) considered it for language learning, and the following discussion is from there.

An *enumeration operator* (see Rogers 1967), Θ, is an algorithmic mapping from SEQ into SEQ such that the following two conditions are satisfied:

- for all $\sigma, \tau \in$ SEQ, if $\sigma \subseteq \tau$, then $\Theta(\sigma) \subseteq \Theta(\tau)$;
- for all texts T, $\lim_{n\to\infty} |\Theta(T[n])| = \infty$.

By extension, we think of Θ as also mapping texts to texts such that $\Theta(T) = \bigcup_n \Theta(T[n])$. Furthermore, we define $\Theta(L) = \{\mathrm{ctnt}(\Theta(T)) : T$ is a text for $L\}$. Intuitively, $\Theta(L)$ denotes the set of languages to whose texts Θ maps texts of L. The reader should note the overloading of this notation because the type of the argument to Θ could be a sequence, a text, or a language.

One says that a sequence of grammars $g_0, g_1, \ldots$ is an acceptable **TxtEx** sequence for L if the sequence of grammars converges to a grammar for L.

$\mathcal{L}_1 \leq_{\mathrm{weak}} \mathcal{L}_2$ if there are two operators Θ and Ψ such that for all $L \in \mathcal{L}_1$, for all texts T for L, $\Theta(T)$ is a text for some $L' \in \mathcal{L}_2$ such that if $g_0, g_1, \ldots$ is an acceptable **TxtEx** sequence for L', then $\Psi(g_0, g_1, \ldots)$ is an acceptable **TxtEx** sequence for L.

Note that different texts for the same language L may be mapped by Θ to texts for different languages in $\mathcal{L}_2$ above. If we require that different texts for L are mapped to texts for the same language L' in $\mathcal{L}_2$, then we get a stronger notion of reduction called strong reduction: $\mathcal{L}_1 \leq_{\mathrm{strong}} \mathcal{L}_2$ if $\mathcal{L}_1 \leq_{\mathrm{weak}} \mathcal{L}_2$ and for all $L \in \mathcal{L}_1$, $\Theta(L)$ contains only one language, where Θ is as in the definition for $\leq_{\mathrm{weak}}$ reduction.

It can be shown that *FIN* is a complete class for **TxtEx** identification with respect to $\leq_{\mathrm{weak}}$ reduction (see Jain and Sharma 1996). Interestingly, it was shown that the class of pattern languages (Angluin 1980), the class $SD = \{L : W_{\min(L)} = L\}$, and the class $COINIT = \{\{x : x \geq n\} : n \in \mathbb{N}\}$ are all equivalent under $\leq_{\mathrm{strong}}$. Let $code$ be a bijective mapping from nonnegative rational numbers to natural numbers. Then, one can show that the class RINIT $= \{\{code(x) : 0 \leq x \leq r, x$ is a rational number$\} : 0 \leq r \leq 1, r$ is a rational number $\}$ is $\leq_{\mathrm{strong}}$ complete for **TxtEx** (see Jain et al. 2001).

Interestingly every finite directed acyclic graph can be embedded into the $\leq_{\mathrm{strong}}$ degree structure (Jain and Sharma 1997a). On the other hand, the degree structure is non-dense in the sense that there exist classes $\mathcal{L}_1$ and $\mathcal{L}_2$ such that $\mathcal{L}_1 <_{\mathrm{strong}} \mathcal{L}_2$, but for any class $\mathcal{L}$ such that $\mathcal{L}_1 \leq_{\mathrm{strong}} \mathcal{L} \leq_{\mathrm{strong}} \mathcal{L}_2$, either $\mathcal{L}_1 \equiv_{\mathrm{strong}} \mathcal{L}$ or $\mathcal{L} \equiv_{\mathrm{strong}} \mathcal{L}_2$. A similar result holds for $\leq_{\mathrm{weak}}$ reducibility (see Jain and Sharma 1997a).

Interesting connections between learning of elementary formal systems (Shinohara 1994), intrinsic complexity, and ordinal mind changes (Freivalds and Smith 1993) were shown in Jain and Sharma (1997b).

Learning Using Oracles

Another method to measure complexity of learning is to see how powerful an oracle (given to the learning machine) has to be to make a class learnable. It can be shown that an oracle A permits to explanatorily learn the class of all recursive functions iff A is high (Adleman and Blum 1991). Furthermore, an oracle is trivial, that is, does not give additional learning power for explanatory learning of function classes iff the oracle has 1-generic Turing degree and is Turing reducible to the halting problem (Slaman and Solovay 1991). The picture is a bit different in the general case of learning languages. For every oracle A, there is an oracle B and a class, which is **TxtEx** learnable using the oracle B but not using the oracle A (Jain and Sharma 1993). Note that there are also classes of languages like Gold's class of all finite languages plus the set of natural numbers which are not **TxtEx** learnable using any oracle. Furthermore, for oracles above the halting problem, **TxtEx** learning and **TxtBc** learning using these oracles coincide.

Acknowledgements Sanjay Jain was supported in part by NUS grant numbers C252-000-087-001, R146-000-

181-112, R252-000-534-112. Frank Stephan was supported in part by NUS grant numbers R146-000-181-112, R252-000-534-112.

Recommended Reading

Adleman L, Blum M (1991) Inductive inference and unsolvability. J Symb Log 56:891–900
Angluin D (1980) Finding patterns common to a set of strings. J Comput Syst Sci 21:46–62
Case J, Smith CH (1983) Comparison of identification criteria for machine inductive inference. Theor Comput Sci 25:193–220
Case J, Jain S, Lange S, Zeugmann T (1999) Incremental concept learning for bounded data mining. Inf Comput 152(1):74–110
Chen K-J (1982) Tradeoffs in inductive inference of nearly minimal sized programs. Inf Control 52:68–86
Daley RP, Smith CH (1986) On the complexity of inductive inference. Inf Control 69:12–40
Freivalds R (1975) Minimal Gödel numbers and their identification in the limit. Lect Not Comput Sci 32:219–225
Freivalds R, Smith CH (1993) On the role of procrastination in machine learning. Inf Comput 107(2):237–271
Freivalds R, Kinber E, Smith CH (1995) On the intrinsic complexity of learning. Inf Comput 123:64–71
Jain S, Sharma A (1993) On the non-existence of maximal inference degrees for language identification. Inf Process Lett 47:81–88
Jain S, Sharma A (1994) Program size restrictions in computational learning. Theor Comput Sci 127:351–386
Jain S, Sharma A (1996) The intrinsic complexity of language identification. J Comput Syst Sci 52:393–402
Jain S, Sharma A (1997a) The structure of intrinsic complexity of learning. J Symb Log 62:1187–1201
Jain S, Sharma A (1997b) Elementary formal systems, intrinsic complexity and procrastination. Inf Comput 132:65–84
Jain S, Kinber E, Wiehagen R (2001) Language learning from texts: degrees of intrinsic complexity and their characterizations. J Comput Syst Sci 63:305–354
Lange S, Zeugmann T (1996) Incremental learning from positive data. J Comput Syst Sci 53:88–103
Pitt L (1989) Inductive inference, DFAs, and computational complexity. In: Analogical and inductive inference, second international workshop (AII 1989). LNAI, vol 397. Springer, Heidelberg, pp 18–44
Rogers H (1967) Theory of recursive functions and effective computability. McGraw-Hill, New York (Reprinted, MIT Press 1987)
Shinohara T (1994) Rich classes inferable from positive data: length–bounded elementary formal systems. Inf Comput 108:175–186
Slaman TA, Solovay R (1991) When oracles do not help. In: Proceedings of the fourth annual workshop on computational learning theory, Santa Cruz. Morgan Kaufmann, pp 379–383
Wiehagen R (1976) Limes-Erkennung rekursiver Funktionen durch spezielle Strategien. J Inf Process Cybern EIK 12:93–99
Wiehagen R (1986) On the complexity of effective program synthesis. In: Jantke K (ed) Analogical and inductive inference. Proceedings of the international workshop. LNCS, vol 265. Springer, pp 209–219

Compositional Coevolution

Synonyms

Cooperative coevolution

Definition

A coevolutionary system constructed to learn composite solutions in which individuals represent different candidate components and must be evaluated together with other individuals in order to form a complete solution. Though not precisely the same as *cooperative coevolution*, there is a significant overlap.

Cross-References

▶ Coevolutionary Learning

Computational Complexity of Learning

Sanjay Jain[1] and Frank Stephan[2]
[1]School of Computing, National University of Singapore, Singapore, Singapore
[2]Department of Mathematics, National University of Singapore, Singapore, Singapore

Definition

Measures of the complexity of learning have been developed for a number of purposes including

▶ Inductive Inference, ▶ PAC Learning, and ▶ Query-Based Learning. The complexity is usually measured by the largest possible usage of resources that can occur during the learning of a member of a class. Depending on the context, one measures the complexity of learning either by a single number/ordinal for the whole class or by a function in a parameter n describing the complexity of the target to be learned. The actual measure can be the number of mind changes, the number of queries submitted to a teacher, the number of wrong conjectures issued, the number of errors made, or the number of examples processed until learning succeeds. In addition to this, one can equip the learner with an oracle and determine the complexity of the oracle needed to perform the learning process. Alternatively, in complexity theory, instead of asking for an NP-complete oracle to learn a certain class, the result can also be turned into the form "this class is unlearnable unless RP = NP" or something similar. (Here RP is the class of decision problems solvable by a randomized polynomial time algorithm, and NP is the class of decision problems solvable by a nondeterministic polynomial time algorithm, and both algorithms never give "yes" answer for an instance of the problem with "no" answer.)

Detail

In ▶ PAC Learning, one usually asks how many examples are needed to learn the concept, where the number of examples needed mainly depends on the Vapnik-Chervonenkis dimension of the class to be learned, the error permitted, and the confidence required. Furthermore, for certain classes of finite Vapnik-Chervonenkis dimension, learnability can still fail when the learner is required to be computable in polynomial time; hence, there is, besides the dimension, also a restriction stemming from the computational complexity of problems such as the complexity of finding concepts consistent with all data observed so far.

For ▶ Query-Based Learning, one common criterion to be looked at is the number of queries made during the learning process. If a class contains 2^n different $\{0,1\}$-valued functions f and one is required to learn the class using membership queries, that is, by asking queries of the form whether $f(x) = 0$ or $f(x) = 1$, then there is a function f on which the learner needs at least n queries until it knows which of the given functions f is; for some classes consisting of 2^n functions, the number of queries needed can be much worse – as much as $2^n - 1$. A well-known result of Angluin is that one can learn the class of all regular languages with polynomially many equivalence and membership queries measured with respect to the number of states of the smallest deterministic finite automaton accepting the language to be learned. Further research has been done dealing with which query algorithms can be implemented by a polynomial time learner and which need for polynomial time learning, in addition to the teacher informing on the target concept, also some oracle supplying information that cannot be computed in polynomial time. See the entry ▶ Query-Based Learning for an overview of these results.

For ▶ Inductive Inference, most complexity measures are measures applying to the overall class and not just a parameterized version. When learning the class of all sets with up to n elements, the learner might first issue the conjecture $\emptyset$ and then revise (up to n times) its hypothesis when a new datum is observed; such a measure is called the mind change complexity of learning. Mind change complexity has been generalized to measure the complexity by recursive ordinals or the notation of these. Furthermore, one can measure the long-term memory of past data observed either by a certain number of examples remembered or by the number of bits stored on a tape describing the long-term memory of the learner. Besides these quantitative notions, a further frequently studied question is the following: Which oracles support the learning process in a way that some classes become learnable using the oracle, but are unlearnable without using any oracle? An example of such a type of result is that the class of all recursive functions can be learned if and only if the learner has access to a high oracle, that is, an oracle that permits to compute a function which dominates (i.e., grows faster than) every recursive function. See the entry ▶ Complexity of Inductive Inference for more information.

Acknowledgements Sanjay Jain was supported in part by NUS grant numbers C252-000-087-001, R146-000-181-112, R252-000-534-112. Frank Stephen was supported in part by NUS grant numbers R146-000-181-112, R252-000-534-112.

Computational Discovery of Quantitative Laws

▶ Equation Discovery

Concept Drift

Claude Sammut[1] and Michael Harries[2]
[1]The University of New South Wales, Sydney, NSW, Australia
[2]Citrix Labs, Advanced Products Group, North Ryde, NSW, Australia

Synonyms

Context-sensitive learning; Learning with hidden context

Definition

Concept drift occurs when the values of hidden variables change over time. That is, there is some unknown context for ▶ concept learning and when that context changes, the learned concept may no longer be valid and must be updated or relearned.

Motivation and Background

Prediction in real-world domains is complicated by potentially unstable phenomena that are not known in advance to the learning system. For example, financial market behavior can change dramatically with changes in contract prices, interest rates, inflation rates, budget announcements, and political and world events. Thus, concept definitions that may have been learned in one context become invalid in a new context. This *concept drift* can be due to changes in context and is often directly reflected by one or more attributes. When changes in context are not reflected by any known attributes they can be said to be *hidden*. Hidden changes in context cause problems for any predictive approach that assumes concept stability.

Structure of the Learning System

Machine learning approaches can be broadly categorized as either ▶ batch Learning or ▶ incremental learning. Batch systems learn offline by examining a large collection of instances *en masse* and form a single concept. Incremental systems evolve and change a concept definition as new observations are processed (Schlimmer and Granger 1986a; Aha et al. 1991; Kolter and Maloof 2003).

The most common approach to learning in domains with hidden changes in context has been to use an incremental learning approach in which the importance of older items is progressively decayed. A popular implementation of this, originally presented in Kubat (1989), is to use a window of recent instances from which concept updates are derived. Other examples of this approach include Widmer and Kubat (1996), Kubat and Widmer (1995), Kilander and Jansson (1993), and Salganicoff (1993). Swift adaptation to changes in context can be achieved by dynamically varying the window size in response to changes in accuracy and concept complexity (Widmer and Kubat 1996).

There are many domains in which the context can be expected not only to change but for earlier contexts to hold again at some time in the future. That is, contexts can repeat in domains such as financial prediction, dynamic control, and underrepresented data mining tasks. In these domains, prediction accuracy can be improved by storing knowledge about past contexts for reuse. FLORA3 (Widmer and Kubat 1993) addresses domains in which contexts recur by storing and retrieving concepts that appear stable as the learner traverses the series of input data.

In many situations, there is no constraint to learn incrementally. For example, many organizations maintain large data bases of historical data that are suitable for data mining. These data bases may hold instances that belong to a number of contexts but do not have this context explicitly recorded. Many of these data bases may incorporate time as an essential attribute, for example, financial records and stock market price data. Interest in mining datasets of this nature suggests the need for systems that can learn global concepts and are sensitive to changing and hidden contexts. Systems such as FLORA3 also imply that an off-line recognition of stable concepts can be useful for ▶ on-line prediction.

An alternative to on-line learning for domains with hidden changes in context is to examine the data *en masse* in an attempt to directly identify concepts associated with stable, hidden contexts. Some potential benefits of such an approach are:

1. Context specific (known as local) concepts could be used as part of a multiple model online predictive system.
2. Local concepts could be verified by experts, or used to improve domain understanding.
3. A model of the hidden context could be induced using context characteristics such as context duration, order, and stability. The model could also use existing attributes and domain feedback if available.
4. Stable contexts identified could be used as target characteristics for selecting additional attributes from the outside world as part of an iterative data mining process.

Splice (Harries et al. 1998) is a ▶ meta-learning system that implements a context sensitive batch learning approach. Splice is designed to identify intervals with stable hidden context, and to induce and refine local concepts associated with hidden contexts.

Identifying Context Change

In many domains with hidden changes in context, time can be used to differentiate hidden contexts. Most machine learning approaches to these domains do not explicitly represent time as they assume that current context can be captured by focusing on recent examples. The implication is that hidden context will be reflected in contiguous intervals of time. For example, an attempt to build a system to predict changes in the stock market could produce the following ▶ decision tree:

$$\begin{array}{l}
\text{Year} \quad > 2002 \\
\qquad \text{Year} < 2005 \\
\qquad\qquad \text{Attribute A} \;=\; \text{true}: \;\text{Market Rising} \\
\qquad\qquad \text{Attribute A} \;=\; \text{false}: \;\text{Market Falling} \\
\qquad \text{Year} \geq 2005 \\
\qquad\qquad \text{Attribute B} \;=\; \text{true}: \;\text{Market Rising} \\
\qquad\qquad \text{Attribute B} \;=\; \text{false}: \;\text{Market Falling}
\end{array}$$

This tree contains embedded knowledge about two intervals of time: in one of which, 2002–2004, attribute A is predictive; in the other, 2005 onward, attribute B is predictive. As time (in this case, year) is a monotonically increasing attribute, future classification using this decision tree will only use attribute B. If this domain can be expected to have recurring hidden context, information about the prior interval of time could be valuable.

The decision tree in the example above contains information about changes in context. We define context as:

> Context is any attribute whose values are largely independent but tend to be stable over contiguous intervals of another attribute known as the environmental attribute.

The ability of decision trees to capture context is associated with the fact that decision tree algorithms use a form of context-sensitive feature selection (CSFS). A number of machine learning algorithms can be regarded as using CSFS including decision tree algorithms (Quinlan 1993), rule induction algorithms (Clark and Niblett 1989), and ▶ ILP systems (Quinlan 1990). All of these systems produce concepts containing local information about context.

When contiguous intervals of time reflect a hidden attribute or context, we call time the environmental attribute. The environmental attribute is not restricted to time alone as it could be any ordinal attribute over which instances of a hidden context are liable to be contiguous. There is also no restriction, in principle, to one dimension. Some alternatives to time as environmental attributes are dimensions of space, and space–time combinations.

Given an environmental attribute, we can utilize a CSFS machine learning algorithm to gain information on likely hidden changes in context. The accuracy of the change points found will be dependent upon at least hidden context duration, the number of different contexts, the complexity of each local concept, and noise.

The CSFS identified context change points can be expected to contain errors of the following types:

1. ▸ Noise or serial correlation errors. These would take the form of additional incorrect change points.
2. Errors due to the repetition of tests on time in different parts of the concept. These would take the form of a group of values clustered around the actual point where the context changed.
3. Errors of omission, change points that are missed altogether.

The initial set of identified context changes can be refined by contextual ▸ clustering.

This process combines similar intervals of the dataset, where the similarity of two intervals is based upon the degree to which a partial model is accurate on both intervals.

Recent Advances

With the increasing amount of data being generated by organizations, recent work on concept drift has focused on mining from high volume ▸ data streams (Hulten et al. 2001; Wang et al. 2003; Kolter and Maloof 2003; Mierswa et al. 2006; Chu and Zaniolo 2004; Gaber et al. 2005). Methods such as Hulten et al.'s, combine decision tree learning with incremental methods for efficient updates, thus avoiding relearning large decision trees. Koltzer and Maloof also use incremental methods combined in an ▸ ensemble.

Cross-References

▸ Decision Tree
▸ Ensemble Learning
▸ Incremental Learning
▸ Inductive Logic Programming
▸ Lazy Learning

Recommended Reading

Aha DW, Kibler D, Albert MK (1991) Instance-based learning algorithms. Mach Learn 6:37–66

Chu F, Zaniolo C (2004) Fast and light boosting for adaptive mining of data streams. In: Advances in knowledge discovery and data mining. Lecture notes in computer science, vol 3056, pp 282–292. Springer, Berlin/New York

Clark P, Niblett T (1989) The CN2 induction algorithm. Mach Learn 3:261–283

Clearwater S, Cheng T-P, Hirsh H (1989) Incremental batch learning. In: Proceedings of the sixth international workshop on machine learning, Ithaca. Morgan Kaufmann, pp 366–370

Domingos P (1997) Context-sensitive feature selection for lazy learners. Artif Intell Rev 11:227–253. [Aha D (ed) Special issue on lazy learning.]

Gaber MM, Zaslavsky A, Krishnaswamy S (2005) Mining data streams: a review. SIGMOD Rec 34(2):18–26

Harries M, Horn K (1996) Learning stable concepts in domains with hidden changes in context. In: Kubat M, Widmer G (eds) Learning in context-sensitive domains (workshop notes). 13th international conference on machine learning, Bari

Harries MB, Sammut C, Horn K (1998) Extracting hidden context. Mach Learn 32(2):101–126

Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. In: KDD'01: proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, pp 97–106

Kilander F, Jansson CG (1993) COBBIT – a control procedure for COBWEB in the presence of concept drift. In: Brazdil PB (ed) European conference on machine learning. Springer, Berlin, pp 244–261

Kolter JZ, Maloof MA (2003) Dynamic weighted majority: a new ensemble method for tracking concept drift. In: Third IEEE international conference on data mining ICDM-2003, Melbourne. IEEE CS Press, pp 123–130
Kubat M (1989) Floating approximation in time-varying knowledge bases. Pattern Recognit Lett 10:223–227
Kubat M (1992) A machine learning based approach to load balancing in computer networks. Cybern Syst J
Kubat M (1996) Second tier for decision trees. In: Machine learning: proceedings of the 13th international conference. Morgan Kaufmann, San Francisco, pp 293–301
Kubat M, Widmer G (1995) Adapting to drift in continuous domains. In: Proceedings of the eighth European conference on machine learning. Springer, Berlin, pp 307–310
Mierswa I, Wurst M, Klinkenberg R, Scholz M, Euler T (2006) Yale: rapid prototyping for complex data mining tasks. In: KDD'06: proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, pp 935–940
Quinlan JR (1990) Learning logical definitions from relations. Mach Learn 5:239–266
Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, San Mateo
Salganicoff M (1993) Density adaptive learning and forgetting. In: Machine learning: proceedings of the tenth international conference. Morgan Kaufmann, San Mateo, pp 276–283
Schlimmer JC, Granger RI Jr (1986a) Beyond incremental processing: tracking concept drift. In: Proceedings AAAI-86. Morgan Kaufmann, Los Altos, pp 502–507
Schlimmer J, Granger R Jr (1986b) Incremental learning from noisy data. Mach Learn 1(3):317–354
Turney PD (1993a) Exploiting context when learning to classify. In: Brazdil PB (ed) European conference on machine learning. Springer, Berlin, pp 402–407
Turney PD (1993b) Robust classification with context sensitive features. In: Paper presented at the industrial and engineering applicatións of artificial intelligence and expert systems, Edinburgh
Turney P, Halasz M (1993) Contextual normalization applied to aircraft gas turbine engine diagnosis. J Appl Intell 3:109–129
Wang H, Fan W, Yu PS, Han J (2003) Mining concept-drifting data streams using ensemble classifiers. In: KDD'03: proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, pp 226–235
Widmer G (1996) Recognition and exploitation of contextual clues via incremental meta-learning. In: Saitta L (ed) Machine learning: proceedings of the 13th international workshop. Morgan Kaufmann, San Francisco, pp 525–533
Widmer G, Kubat M (1993) Effective learning in dynamic environments by explicit concept tracking. In: Brazdil PB (ed) European conference on machine learning. Springer, Berlin, pp 227–243
Widmer G, Kubat M (1996) Learning in the presence of concept drift and hidden contexts. Mach Learn 23:69–101

Concept Learning

Claude Sammut
The University of New South Wales, Sydney,
NSW, Australia

Synonyms

Categorization; Classification learning

Definition

The term *concept learning* is originated in psychology, where it refers to the human ability to learn categories for object and to recognize new instances of those categories. In machine learning, concept is more formally defined as "inferring a boolean-valued function from training examples of its inputs and outputs" (Mitchell 1997).

Background

Bruner et al. (1956) published their book *A Study of Thinking*, which became a landmark in psychology and would later have a major impact on machine learning. The experiments reported by Bruner, Goodnow, and Austin were directed toward understanding a human's ability to categorize and how categories are learned.

> We begin with what seems a paradox. The world of experience of any normal man is composed of a tremendous array of discriminably different objects, events, people, impressions... But were we to utilize fully our capacity for registering the differences in things and to respond to each event encountered as unique, we would soon be overwhelmed by the complexity of our

> environment…The resolution of this seeming paradox…is achieved by man's capacity to categorize. To categorize is to render discriminably different things equivalent, to group objects and events and people around us into classes…The process of categorizing involves…an act of invention…If we have learned the class "house" as a concept, new exemplars can be readily recognised. The category becomes a tool for further use. The learning and utilization of categories represents one of the most elementary and general forms of cognition by which man adjusts to his environment.

The first question that they had to deal with was that of representation: what is a concept? They assumed that objects and events could be described by a set of attributes and were concerned with how inferences could be drawn from attributes to class membership. Categories were considered to be of three types: conjunctive, disjunctive, and relational.

> …when one learns to categorize a subset of events in a certain way, one is doing more than simply learning to recognise instances encountered. One is also learning a rule that may be applied to new instances. The concept or category is basically, this "rule of grouping" and it is such rules that one constructs in forming and attaining concepts.

The notion of a rule as an abstract representation of a concept influenced research in machine learning. For example, ▸ decision tree learning was used as a means of creating a cognitive model of concept learning (Hunt et al. 1966). This model later inspired Quinlan's development of ID3 (Quinlan 1983).

The learning experience may be in the form of examples from a trainer or the results of trial and error. In either case, the program must be able to represent its observations of the world, and it must also be able to represent hypotheses about the patterns it may find in those observations. Thus, we will often refer to the ▸ observation language and the ▸ hypothesis language. The observation language describes the inputs and outputs of the program and the hypothesis language describes the internal state of the learning program, which corresponds to its theory of the concepts or patterns that exist in the data.

The input to a learning program consists of descriptions of objects from the universe and, in the case of ▸ supervised learning, an output value associated with the example. The universe can be an abstract one, such as the set of all natural numbers, or the universe may be a subset of the real world. No matter which method of representation we choose, descriptions of objects in the real world must ultimately rely on measurements of some properties of those objects. These may be physical properties such as size, weight, and color or they may be defined for objects, for example, the length of time a person has been employed for the purpose of approving a loan. The accuracy and reliability of a learned concept depends on the accuracy and reliability of the measurements.

A program is limited in the concepts that it can learn by the representational capabilities of both observation and hypothesis languages. For example, if an attribute/value list is used to represent examples for an induction program, the measurement of certain attributes and not others clearly places bounds on the kinds of patterns that the learner can find. The learner is said to be *biased* by its observation language (see ▸ Language Bias). The hypothesis language also places constraints on what may and may not be learned. For example, in the language of attributes and values, relationships between objects are difficult to represent. Whereas, a more expressive language, such as first-order logic, can easily be used to describe relationships.

Unfortunately, representational power comes at a price. Learning can be viewed as a search through the space of all sentences in a language for a sentence that best describes the data. The richer the language, the larger is the search space. When the search space is small, it is possible to use "brute force" search methods. If the search space is very large, additional knowledge is required to reduce the search.

Rules, Relations, and Background Knowledge

In the early 1960s, there was no discipline called "machine learning." Instead, learning was considered to be part of "pattern recognition," which had not yet split from AI. One of the main problems addressed at that time was how to represent patterns so that they could be recognized easily. Symbolic description languages were developed to be expressive and learnable.

Banerji (1960, 1962) first devised a language, which he called a "description list," which utilized an object's attributes to perform pattern recognition. Pennypacker, a masters student of Banerji at the Case Institute of Technology, implemented the recognition procedure and also used Bruner, Goodnow, and Austin's *Conservative Focussing Strategy* to learn conjunctive concepts (Pennypacker 1963). Bruner, Goodnow, and Austin describe the strategy as follows:

> ...this strategy may be described as finding a positive instance to serve as a focus, then making a sequence of choices each of which alters but one attribute value [of the focus] and testing to see whether the change yields a positive or negative instance. Those attributes of the focus which, when changed, still yield positive instance are *not* part of the concept. Those attributes of the focus that yield negative instances when changed *are* features of the concept.

The strategy is only capable of learning *conjunctive concepts*, that is, the concept description can only consist of a simple conjunction of tests on attribute values. Recognizing the limitations of simple attribute/value representations, Banerji (1964) introduced the use of predicate logic as a description language. Thus, Banerji was one of the earliest advocates of what would, many years later, become *Inductive Logic Programming*.

In the 1970s, a series of algorithms emerged that developed concept learning further. Winston's ARCH program (Winston, Learning structural descriptions from examples. PhD Thesis, MIT Artificial Intelligence Laboratory, 1970, unpublished) was influential as one of the first widely known concept learning programs. Michalski (1973, 1983) devised the Aq family of learning algorithms that set some of the early benchmarks for learning programs. Early relational learning programs were developed by Hayes-Roth (1973), Hayes-Roth and McDermott (1977), and Vere (1975, 1977).

Banerji emphasized the importance of a description language that could "grow." That is, its descriptive power should increase as new concepts are learned. These concepts become background knowledge for future learning. A simple example from Banerji (1980) illustrates the use of background knowledge. There is a language for describing instances of a concept and another for describing concepts. Suppose we wish to represent the binary number, 10, by a left-recursive binary tree of digits "0" and "1":

$$[\text{head}: \ [\text{head}: \ 1; \ \text{tail}: \ \text{nil}]; \text{tail}: 0]$$

"head" and "tail" are the names of attributes. Their values follow the colon. The concepts of binary digit and binary number are defined as

$$\begin{aligned} x \in digit &\equiv x = 0 \vee x = 1 \\ x \in num &\equiv (tail(x) \in digit \\ &\quad \wedge head(x) = nil) \\ &\quad \vee (tail(x) \in digit \\ &\quad \wedge head(x) \in num) \end{aligned}$$

Thus, an object belongs to a particular class or concept if it satisfies the logical expression in the body of the description. Note that the concept above is *disjunctive*. Predicates in the expression may test the membership of an object in a previously learned concept and can express *relations* between objects. Cohen and Sammut (1982) devised a learning system based on Banerji's ideas of a growing concept description language and this was further extended by Sammut and Banerji (1986).

Concept Learning and Noise

One of the most severe drawbacks of early concept learning systems was that they assumed that data sets were not noisy. That is, all attribute values and class labels in the training data are assumed to be correct. This is unrealistic in most real applications. Thus, concept learning systems began incorporating statistical measures to minimize the effects of noise and to estimate error rates (Breiman et al. 1984; Cohen 1995; Quinlan 1986, 1993).

Learning to classify objects from training examples has gone on to become one of the central themes of machine learning research. As the robustness of classification systems has increased, they have found many applications, particularly in data mining but in a broad range of other areas.

Cross-References

- ▶ Data mining on Text
- ▶ Decision Tree
- ▶ Induction
- ▶ Inductive Logic Programming
- ▶ Learning as Search
- ▶ Relational Learning
- ▶ Rule Learning

Recommended Reading

Banerji RB (1960) An information processing program for object recognition. Gen Syst 5:117–127

Banerji RB (1962) The description list of concepts. Commun Assoc Comput Mach 5(8):426–432

Banerji RB (1964) A language for the description of concepts. Gen Syst 9:135–141

Banerji RB (1980) Artificial intelligence: a theoretical approach. North Holland, New York

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. Wadsworth, Belmont

Bruner JS, Goodnow JJ, Austin GA (1956) A study of thinking. Wiley, New York

Cohen WW (1995) In fast effective rule induction. In: Proceedings of the twelfth international conference on machine learning, Lake Tahoe. Morgan Kaufmann, Menlo Park

Cohen BL, Sammut CA (1982) Object recognition and concept learning with CONFUCIUS. Pattern Recognit J 15(4):309–316

Hayes-Roth F (1973) A structural approach to pattern learning and the acquisition of classificatory power. In: First international joint conference on pattern recognition, Washington, DC, pp 343–355

Hayes-Roth F, McDermott J (1977) Knowledge acquisition from structural descriptions. In: Fifth international joint conference on artificial intelligence, Cambridge, MA, pp 356–362

Hunt EB, Marin J, Stone PJ (1966) Experiments in induction. Academic, New York

Michalski RS (1973) Discovering classification rules using variable valued logic system VL1. In: Third international joint conference on artificial intelligence, Stanford, pp 162–172

Michalski RS (1983) A theory and methodology of inductive learning. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning: an artificial intelligence approach. Tioga, Palo Alto

Mitchell TM (1997) Machine learning. McGraw-Hill, New York

Pennypacker JC (1963) An elementary information processor for object recognition. SRC No. 30-I-63-1. Case Institute of Technology

Quinlan JR (1983) Learning efficient classification procedures and their application to chess end games. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning: an artificial intelligence approach. Tioga, Palo Alto

Quinlan JR (1986) The effect of noise on concept learning. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning: an artificial intelligence approach, vol 2. Morgan Kaufmann, Los Altos

Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, San Mateo

Sammut CA, Banerji RB (1986) Learning concepts by asking questions. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning: an artificial intelligence approach, vol 2. Morgan-Kaufmann, Los Altos, pp 167–192

Vere S (1975) Induction of concepts in the predicate calculus. In: Fourth international joint conference on artificial intelligence, Tbilisi, pp 351–356

Vere SA (1977) Induction of relational productions in the presence of background information. In: Fifth international joint conference on artificial intelligence, Cambridge, MA

Conditional Random Field

A *Conditional Random Field* is a form of ▶ Graphical Model for segmenting and ▶ classifying sequential data. It is the ▶ discriminative learning counterpart to the ▶ generative learning Markov Chain model.

Recommended Reading

Lafferty J, McCallum A, Pereira F (2001) Conditional random fields: probabilistic models for segmenting and labeling sequence data. In: Proceedings of the 18th international conference on machine learning. Morgan Kaufmann, San Francisco, pp 282–289

Confirmation Theory

The branch of philosophy concerned with how (and indeed whether) evidence can confirm a hypothesis, even though typically it does not entail it. A distinction is sometimes drawn between *total confirmation*: how well confirmed a hypothesis is, given all available evidence and *weight-of-evidence*: the amount of extra confirmation added to the total confirmation of a hypothesis by a particular piece of evidence. Confirmation is often measured by the probability of a hypothesis conditional on evidence.

Confusion Matrix

Kai Ming Ting
Federation University, Mount Helen, VIC, Australia

Definition

A confusion matrix summarizes the classification performance of a ▶ classifier with respect to some ▶ test data. It is a two-dimensional matrix, indexed in one dimension by the true class of an object and in the other by the class that the classifier assigns. Table 1 presents an example of confusion matrix for a three-class classification task, with the classes A, B, and C.

The first row of the matrix indicates that 13 objects belong to the class A and that 10 are correctly classified as belonging to A, two misclassified as belonging to B, and one as belonging to C.

Confusion Matrix, Table 1 An example of a three-class confusion matrix

Actual class		Assigned class		
		A	*B*	*C*
	A	10	2	1
	B	0	6	1
	C	0	3	8

Confusion Matrix, Table 2 The outcomes of classification into positive and negative classes

Actual class		Assigned class	
		Positive	Negative
	Positive	TP	FN
	Negative	FP	TN

A special case of the confusion matrix is often utilized with two classes, one designated the *positive* class and the other the *negative* class. In this context, the four cells of the matrix are designated as ▶ *true positives* (TP), ▶ *false positives* (FP), ▶ *true negatives* (TN), and ▶ *false negatives* (FN), as indicated in Table 2.

A number of measures of classification performance are defined in terms of these four classification outcomes.

▶ *Specificity* = ▶ *True negative rate* = TN/(TN + FP)

▶ *Sensitivity* = ▶ *True positive rate* = ▶ *Recall* = TP/ (TP + FN)

▶ *Positive predictive value* = ▶ *Precision* = TP/(TP + FP)

▶ *Negative predictive value* = TN/(TN + FN)

Conjunctive Normal Form

Bernhard Pfahringer
University of Waikato, Hamilton, New Zealand

Conjunctive normal form (CNF) is an important normal form for propositional logic. A logic formula is in conjunctive normal form if it is a single conjunction of disjunctions of (possibly negated) literals. No more nesting and no other negations are allowed. Examples are:

$$a$$
$$\neg b$$
$$a \wedge b$$
$$(a \vee \neg b) \wedge (c \vee d)$$
$$\neg a \wedge (b \vee \neg c \vee d) \wedge (a \vee \neg d)$$

Any arbitrary formula in propositional logic can be transformed into conjunctive normal form by application of the laws of distribution, De Morgan's laws, and by removing double negations. It is important to note that this process can lead to exponentially larger formulas which implies that the process in the worst case runs in exponential time. An example for this behavior is the following formula given in ▸ disjunctive normal form (DNF), which is linear in the number of propositional variables in this form. When transformed into conjunctive normal form (CNF), its size is exponentially larger.

DNF: $(a_0 \wedge a_1) \vee (a_2 \wedge a_3) \vee \ldots \vee (a_{2n} \wedge a_{2n+1})$
CNF: $(a_0 \vee a_2 \vee \ldots \vee a_{2n}) \wedge (a_1 \vee a_2 \vee \ldots \vee a_{2n}) \wedge \ldots \wedge (a_1 \vee a_3 \vee \ldots \vee a_{2n+1})$

Recommended Reading

Russell S, Norvig P (2002) Artificial intelligence: a modern approach. Prentice Hall, p 215

Connection Strength

▸ Weight

Connections Between Inductive Inference and Machine Learning

John Case[1] and Sanjay Jain[2]
[1]University of Delaware, Newark, DE, USA
[2]School of Computing, National University of Singapore, Singapore, Singapore

Abstract

Inductive inference is a branch of computational learning theory which deals with learning in the limit. Though this topic deals with mostly theoretical work, it has provided some results which can be of use to practical machine learning. Some of these works include the work multitask or context-sensitive learning, learnability of elementary formal systems, behavioral cloning, learning to coordinate, geometrical clustering, and so on. The results in these areas also often give insights into limitations of science.

Definition

Inductive inference is a theoretical framework to model learning in the limit. Here we will discuss some results in inductive inference, which have relevance to machine learning community.

The mathematical/theoretical area called ▸ inductive inference is also known as *computability theoretic learning* and *learning in the limit* (Jain et al. 1999; Odifreddi 1999) typically *but, as will be seen below, not always* involves a situation depicted in (1) just below.

$$\text{Data}\quad d_0, d_1, d_2, \ldots \xrightarrow{\text{In}} M \xrightarrow{\text{Out}} \text{Programs}\quad e_0, e_1, e_2, \ldots . \tag{1}$$

Let $\mathbb{N}$ = the set of nonnegative integers. Strings, including program strings, computer reals, and other data structures, inside computers, are finite bit strings and, hence, can be coded into $\mathbb{N}$. Therefore, mathematically at least, it is without loss of mathematical generality that we sometimes use the data type $\mathbb{N}$ where standard practice would use a different type.

In (1), $d_0, d_1, d_2, \ldots$ can be, e.g., the successive values of a function $f : \mathbb{N} \rightarrow \mathbb{N}$ or the elements of a (formal) language $L \subseteq \mathbb{N}$ in some order, M is a machine, the e_i's are from some hypothesis space of programs, and, for M's *successful* learning, later e_i's exactly or approximately compute the f or L.

Such learning is *offline*: in successful cases, one comes away with programs for past and future data. For the related problem of *online extrapolation* of next values for a function f, *suitable* e_i's may *be* the values of $f(i)$'s based on having seen *strictly* prior values of f.

Detail

We will discuss the off-line case until we say otherwise. It is typical in applied machine learning to present to a learner whatever data one has and to obtain *one* corresponding program hopefully for predicting these data and future data. In inductive inference the case where only one program is output is called *one-shot* learning. More typically, in inductive inference, one allows for *mind changes*, i.e., for a succession of output programs, as one receives successively more input data, with the later programs hopefully eventually being useful for predictions. Typically, one does not get success on one's first conjecture/output program, but, rather, one may achieve success eventually or, as it is said, *in the limit* after some sequence of trial and error. It is helpful at this juncture to present a problem for which this latter approach makes more sense than the one-shot approach.

We will consider some different criteria of successful learning of f or L by M. For example, **Ex**-style criteria will require that all but finitely many of the e_i's are *syntactically* the same *and* do a reasonable job of computing the f or L. **Bc**-style criteria are more relaxed, more powerful, but less useful (Bārzdiņš 1974; Case and Lynes 1982; Case and Smith 1983): they do not require almost all e_i's be the same syntactically.

Here is a well-known regression technique from, e.g., Hildebrand (1956), for exactly "curve-fitting" polynomials. It is the method involving calculating *forward differences*. We express it as a learning machine M_0 and illustrate with its being fed an *example* data sequence generated by a cubic polynomial

$$x^3 - 2x^2 + 2x + 3. \qquad (2)$$

Connections Between Inductive Inference and Machine Learning, Table 1 Example sequence and its iterated forward differences

Sequence:	3		4		7		18		43
1st Diffs:		1		3		11		25	
2nd Diffs:			2		8		14		
3rd Diffs:				6		6			

See Hildebrand (1956), for how to recover the polynomials themselves.

M_0, fed a finite data sequence of natural numbers, first looks for iterated forward differences to become (apparently) constant and then outputs a rule/program, which uses the (apparent) constant to extrapolate the data sequence for any desired prediction. For example, were M_0 given the data sequence in the top row of Table 1, it would calculate 6 to be the apparent constant after *three* differencings, so M_0 then outputs the following informal rule/program.

> ► To generate the level 0 sequence, at level 0, start with 3; at level 1, start with 1; at level 2, start with 2; add the apparent constant 6 from level 3 to get successive level 2 data items; add successive level 2 items to get successive level 1 data items; *finally*, add successive level 1 items to get as many successive level 0 data items as needed for prediction.

This program, eventually output by M_0 when its input the whole top row of Table 1, correctly predicts the elements of the cubic polynomial, on successive values in $\mathbb{N}$ – the whole sequence $3, 4, 7, 18, 43, 88, 159, \ldots$. Along the way, though, just after the first data point, M_0 thinks the apparent constant is 0; just after the second that it is 1; just after the third that it is 2; and only after more of the data points does it converge for this cubic polynomial to the apparent (and, on *this* example, actual) constant 6. In general, M_0, on a polynomial of degree m, *changes its mind* up to m times until converging to its final program (of course on $f(x) = 2^x$, M_0 never converges, and each level of forward differences is just the sequence f again.).

Hence, M_0 above **Ex**-learns, e.g., the integer polynomials $f : N \rightarrow N$, but it does not in general one-shot learn these polynomials – since the data alone do not disclose the degree of a generating polynomial.

In this entry we survey some results from inductive inference but with an eye to topics having something to say regarding or to applied machine learning. In some cases, the theoretical results lend mathematical support to preexisting empirical observations about the efficacy of known machine learning techniques. In other cases, the theoretical results provide some, typically abstract, suggestions for the machine learning practitioner. In some of these cases, some of the suggestions apparently pay off in others, intriguingly, we do not know yet.

Multitask or Context-Sensitive Learning

In empirical, applied machine learning, *multitask* or *context-sensitive learning* involves trying to learn Y by first (de Garis 1990a,b; Fahlman 1991; Thrun 1996; Thrun and Sullivan 1996; Tsung and Cottrell 1989; Waibel 1989a,b) *or* simultaneously (Caruana 1993, 1996; Matwin and Kubat 1996; Bartlmae et al. 1997; Dietterich et al. 1995; Mitchell et al. 1994; Pratt et al. 1991; Sejnowski and Rosenberg 1986) trying to learn also X – even in cases where there may be no inherent interest in learning X. There is, in many cases, an apparent *empirical* advantage in doing this for some X, Y. It can happen that Y is not apparently or easily learnable by itself but is learnable if one learns X first or simultaneously in some case X itself can be a sequence of tasks $X_1, \ldots, X_n$. Here the X_is may need to be learned sequentially or simultaneously to learn Y. For example, to teach a robot to drive a car, it is useful to train it also to predict the center of the road markings (see, e.g., Baluja and Pomerleau 1995; Caruana 1996). For another example, an experimental system to predict the value of German *Daimler* stock performed better when it was modified to track simultaneously the German stock index DAX (Bartlmae et al. 1997). The value of the Daimler stock here was the primary or target concept and the value of the DAX – a related concept – provided useful auxiliary context.

Angluin et al. (1989) shows *mathematically* that, in effect, there are (mathematical) learning scenarios for which it was *provable* that Y could not be learned without learning X *first* , and, in other scenarios (Angluin et al. 1989; Kinber et al. 1995), Y could not be learned without *simultaneously* learning X. These *mathematical* results provide a kind of evidence that the *empirical* observations as to the *apparent* usefulness of multitask or context-sensitive learning *may* not be illusionary, luck, or a mere accident of happening to use some data sets but not others.

For illustration, here is a particularly simple theoretical example needing to be learned *simultaneously* and similar to examples in Angluin et al. (1989). Let $\mathcal{R}$ be the set of all computable functions mapping $\mathbb{N}$ to $\mathbb{N}$. We use numerical names in $\mathbb{N}$ for programs. Let

$$\mathcal{S} = \{(f, g) \in \mathcal{R} \times \mathcal{R} \mid f(0) \text{ is a program for } g \wedge g(0) \text{ is a program for } f\}. \tag{3}$$

We say (p, q) is a program for $(f, g) \in \mathcal{R} \times \mathcal{R}$ iff p is a program for f and q is a program for g.

Consider a machine M which, if, as in (1), M is fed $d_0, d_1, \ldots$, but *where* each d_i is $(f(i), g(i))$, then M outputs each $e_i = (g(0), f(0))$. Clearly, M one-shot learns $\mathcal{S}$. It can be easily shown that the component f's and g's *for* $(f, g) \in \mathcal{S}$ are not *separately* even **Bc**-learnable. It is important to note that, perhaps quite unlike real-world problems, the definition of this example $\mathcal{S}$ employs a simple self-referential coding trick: useful programs are coded into values of the functions at argument zero. A number of inductive inference results have been proved by means of (sometimes more complicated) self-referential coding tricks (see, e.g., Case 1994). Bārzdiņš indirectly (see Zeugmann 1986) provided a kind of informal robustness idea in his attempt to be rid of such coding tricks in inductive inference. More formally, Fulk (1990) considered a learnability result involving a witnessing class $\mathcal{C}$ of (tuples of) functions to be *robust* iff each computable scrambling of $\mathcal{C}$ also witnesses the learnability result (the allowed computable scramblers are the *general recursive operators* of (Rogers 1967), but we omit the formal details herein.) Example: A simple shift scrambler converting each f to f', where $f'(x) = f(x + 1)$, would eliminate the coding tricks just above – since the values of f at argument zero would be lost in this scrambling. Some inductive inference results hold robustly and some not (see, e.g., Fulk 1990; Jain et al. 1999, 2001; Jain 1999;

Case et al. 2000). Happily, the $\mathcal{S} \subseteq \mathcal{R} \times \mathcal{R}$ above (i.e., learnable, but its components not) can be replaced by a more complicated class $\mathcal{S}'$ that *robustly* witnesses the same result. This is better *theoretical* evidence that the empirically noticed efficacy of multitask or context-sensitive learning is not just an accident. It is residually important to note that (Jain et al. 2001) shows, though, that the computable scramblers cannot get rid of more sophisticated coding tricks they called topological. $\mathcal{S}'$ mentioned just above turns out to *employ* this latter kind of coding trick. It is hypothesized in Case et al. (2000) that nature likely employs some sophisticated coding tricks itself. For a separate informal argument about coding tricks of nature, see Case (1999). Ott and Stephan (2002) introduce a finite invariance constraint on top of robustness. This so-called hyperrobustness does destroy all coding tricks, and the result about the *theoretical* efficacy of multitask or context-sensitive learning is *not* hyperrobust. However, hyperrobustness, perhaps, leaves unrealistically sparse structure.

Final note: Machine learning is an engineering endeavor. However, philosophers of science as well as practitioners in classical scientific disciplines should likely be considering the relevance of multitask or context-sensitive inductive inference to their endeavors.

Special Cases of Inductive Logic Programming

In this section we discuss some *learning in the limit* results for *elementary formal systems (EFSs)* (Smullyan 1961). Essentially, EFSs are programs in a string rewriting system. It is well known (Arikawa et al. 1992) that EFSs are essentially (pure) logic programs over strings. Hence, the results have possible relevance for ▶ inductive logic programming (ILP) (Muggleton and De Raedt 1994; Lavrač and Džeroski 1994; Bratko and Muggleton 1995; Mitchell 1997).

First we will discuss some important special cases based on Angulin's *pattern languages* (Angluin 1980).

A *pattern language* is (by definition) one generated by all the positive length substitution instances in a *pattern*, such as,

$$abXYcbbZXa \tag{4}$$

— where the variables (for substitutions) are depicted in upper case and the constants/terminals in lower case and are from, say, the alphabet {a,b,c}. Just below is an EFS or logic program based on this example pattern.

$$abXYcbbZXa \leftarrow . \tag{5}$$

It must be understood, though, that in (5) and in the next example EFS below, only positive length strings are allowed to be substituted for the variables.

Angluin (1980) showed the **Ex**-learnability of the class of pattern languages from positive data. For these results, in the paradigm of (1) above $d_0, d_1, d_2, \ldots$ is a listing or presentation of some formal language L over a finite nonempty alphabet, and the e_i's are programs that generate languages. In particular, for Angluin's M, for L a pattern language, the e_i's are patterns, and, for each presentation of L, all but finitely many of the corresponding e_i's are the same *correct* pattern for L.

Much work has been done on the learnability of pattern languages, e.g., Salomaa (1994a,b), Case et al. (2001), and on bounded finite unions thereof, e.g., Shinohara (1983), Wright (1989), Kilpeläinen et al. (1995), Brazma et al. (1996), and Case et al. (1999).

Regarding bounded finite unions of pattern languages, an *n-pattern language* is the union of the pattern languages for some n patterns $P_1, \ldots, P_n$. Each n-pattern language is also **Ex**-learnable from positive data (see Wright 1989). An EFS or logic program corresponding to the n-patterns $P_1, \ldots, P_n$ and generating the corresponding n-pattern language is just below.

$$\begin{gathered} P_1 \leftarrow . \\ \vdots \\ P_n \leftarrow . \end{gathered}$$

Pattern language learning algorithms have been successfully applied toward some problems in molecular biology; see, e.g., Shimozono et al. (1994), Shinohara and Arikawa (1995).

Lange and Wiehagen (1991) present an interesting *iterative* (Wiehagen 1976) algorithm learning the class of pattern languages – from positive data only and with fair polynomial time constraints (for examples of fair vs. unfair polynomial time learning, see Case and Kötzing 2009). *Iterative learners* are **Ex**-learners for which each output depends only on its just prior output (if any) and the input data element currently seen. Their algorithm works in polynomial time (actually quadratic time) in the length of the latest data item and the previous hypothesis. Furthermore, the algorithm has a linear set of good examples, in the sense that if the input data contains these good examples, then the algorithm already converges to the correct hypothesis. The number of good examples needed is at most $|P| + 1$, where P is a pattern generating the data $d_0, d_1, d_2, \ldots$ for the language being learned. This algorithm may be useful in practice due to its fast run time and being able to converge quickly, *if* enough good data is available early. Furthermore, due to iterativeness, it does not need to store previous data!

Zeugmann (1998) considers *total* learning time up to convergence of the algorithm just discussed in the just prior paragraph. Note that, for arbitrary presentations, $d_0, d_1, d_2, \ldots$, of a pattern language, this time can be unbounded. In the best case, it is polynomial in the length of a generating pattern P, where $d_0, d_1, d_2, \ldots$ is based on using P to get good examples early – in fact the time taken in the best case is $\Theta(|P|^2 log_s(s + k))$, where P is the pattern, s is the alphabet size, and k is the number of variables in P. Much more interesting is the case of *average time* taken up to convergence. The probability distribution (called *uniform* by Zeugmann) considered is as follows. A variable X is replaced by a string w with probability $\frac{1}{(2s)^{|w|}}$ (i.e., all strings of length r together have probability 2^{-r}, and the distribution is uniform among strings of length r). Different variables are replaced independently of each other. In this case the average total time up to convergence is $O(2^k k^2 s|P|^2 log_s(ks))$. The main thing is that for average case on probabilistic data (as can be expected in real life, though not necessarily with this kind of uniform distribution), the algorithm converges pretty fast and computations are done efficiently.

A number of papers consider **Ex**-learning of EFSs (Krishna Rao 1996; Krishna Rao and Sattar 1998; Krishna Rao 2000, 2004, 2005) including with various bounds on the number of mind changes until syntactic convergence to correct programs (Jain and Sharma 1997, 2002). The EFSs considered are patterns, n-patterns, those with a constant bound on the length of clauses, and some with constant bounds on search trees. The mind change bounds are typically more dynamic than those given by constants: they involve counting down from finite representations (called *notations*) for infinite constructive ordinals. *An example* of this kind of bound: one can algorithmically, based on some input parameters, decide how many mind changes will be allowed. In *other* examples, the decision as to how many mind changes will be allowed can be algorithmically revised some constant number of times. It is possible that not yet created special cases of some of these algorithms could be made feasible enough for practice.

Learning Drifting Concepts

A drifting concept to be learned is one which is a moving target (see ▸ Concept Drift). In some machine learning applications, concept drift must be dealt with (Bartlett et al. 1996; Blum and Chalasani 1992; Devaney and Ram 1994; Freund and Mansour 1997; Helmbold and Long 1994; Kubat 1992; Wrobel 1994; Widmer and Kubat 1996). An inductive inference contribution is (Case et al. 2001) in which it is shown, for online *extrapolation* by computable martingale betting strategies, upper bounds on the "speed" of the moving target that permit success at all. Here success is to make unbounded amounts of "money" betting on correctness of one's extrapolations. Here is an illustrative result from

(Case et al. 2001). For the pattern languages considered in the previous section, only *positive* length strings of terminals can be substituted for a variable in an associated pattern. The (difficult to learn) *pattern languages with erasing* are just the languages obtained by also allowing the substitution of the empty string for variables in a pattern. For our example, we restrict the terminal alphabet to be {0,1}. With each pattern language with erasing L (over this terminal alphabet), we associate its characteristic function χ_L, which is 1 on terminal strings in L and 0 on those not in L. For ε denoting the empty string, and for the terminal strings in length-lexicographical order, $\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$, we would input a χ_L itself to a potential extrapolating machine as the bit string, $\chi_L(\varepsilon), \chi_L(0), \chi_L(1), \chi_L(00), \chi_L(01), \ldots$. Let $\mathcal{E}$ be the class of these characteristic functions. Pick a positive integer constant p. To model *drift with permanence* p, we imagine that a potential extrapolator for $\mathcal{E}$ receives successive bits from a member of $\mathcal{E}$ but keeps switching to the next bits of another, etc., *but* it must see at least p bits in a row of each member of $\mathcal{E}$ it sees before it can see the next bits of another. p is, then, a speed limit on drift. The result is that some suitably clever computable martingale betting strategy is successful at extrapolating $\mathcal{E}$ with drift permanence (speed limit on drift) of $p = 7$.

Behavioral Cloning

Kummer and Ott (1996) and Case et al. (2002) studied learning in the limit of winning control strategies for *closed computable games*. These games nicely model *reactive* process-control problems. Included are such example process-control games as regulating temperature of a room to be in a desired interval, forever after no more than some *fixed* number of moves between the thermostat and processes disturbing the temperature (Roughly, *closed computable games* are those so that one can tell algorithmically when one has lost. A temperature control game that requires stability forever after some *undetermined* finite number of moves is *not* a closed computable game. For a more formal treatment, see Cenzer and Remmel (1992), Maler et al. (1995), Thomas (1995), and Kummer and Ott (1996).

In machine learning, there are cases where one wants to teach a machine some motor skill possessed by human experts and where these human experts do not have access to verbalizable knowledge about how they perform expertly. Piloting an aircraft or expert operation of a swinging shipyard crane provide examples, and machine learning employs, in these cases, ▸ *behavioral cloning*, which uses direct performance data from the experts (Bain and Sammut 1999; Bratko et al. 1998; Šuc 2003).

Case et al. (2002) study the effects on learning in the limit *closed computable games* where the learning procedures also had access to the *behavioral performance* (but not the algorithms) of masters/experts at the games. For example, it is showed that, in some cases, there is better performance cloning $n + 1$ disparate masters over cloning only n. For a while it was not known *in machine learning* how to clone multiple experts even after Case et al. (2002) was known to some; however, independently of Case et al. (2002), and later, Dorian Šuc (2003) found a way to clone behaviorally more than one human expert simultaneously (for the free-swinging shipyard crane problem) – by having more than one level of feedback control, *and* he got enhanced performance from cloning the multiple experts!

Learning to Coordinate

Montagna and Osherson (1999) begin the study of learning in the limit to *coordinate* (digital) moves between at least two agents.

The machines of Montagna and Osherson (1999) are, in effect, general extrapolating devices (Montagna and Osherson 1999; Case et al. 2005). Technically, and without loss of generality of the results, we restrict the moves of each coordinator to bits, i.e., zeros and ones. *Coordination* is achieved between two coordinators iff each, reacting to the bit sequence of the other, eventually (in the limit) matches it bit for bit. Montagna and Osherson (1999) give an example of two people who show up in a park each day at one of noon (bit 0) or 6pm (bit 1); each *silently* watches the other's past behavior, and each *tries*, based on the past behavior of the other, to show up eventually exactly when the other shows up. If they manage it, they have learned to coordinate.

A *blind* coordinator is one that reacts only to the *presence* of a bit from another process, *not* to *which* bit the other process has played (Montagna and Osherson 1999).

Case et al. (2005) developed and studied the notion of probabilistically correct algorithmic coordinators. Next is a sample of theorems to the effect that just a few random bits can enhance learning to coordinate.

Theorem 1 (Case et al. 2005) *Suppose* $0 \leq p < 1$. *There exists a class of deterministic algorithmic coordinators* $\mathcal{C}$ *such that:*

(1) No *deterministic algorithmic coordinator can coordinate with all of* $\mathcal{C}$
(2) For k *chosen so that* $1-2^{-k} \geq p$, *there exists a blind,* probabilistic *algorithmic coordinator* **PM**, *such that:*
  (i) For each member of $\mathcal{C}$, **PM** *can coordinate with with probability* $1-2^{-k} \geq p$
  (ii) **PM** *is* k-memory limited *in the sense of (Osherson et al. 1986, P. 66); more specifically,* **PM** *needs to remember whether it is outputting one of its first* k *bits* — which are its only random bits *(e.g., for* $p = \frac{255}{256}$, *a mere* $k = 8$ *random bits suffice.).*

Regarding possible eventual applicability: Maye et al. (2007) cite finding deterministic chaos but *not* randomness in the behavior of *animals*. Hence, animals may not be exploiting random bits in learning anything, including to coordinate. However, one might build artifactual devices to exploit randomness, say, from radioactive decay, including, then, for enhancing learning to coordinate.

Learning Geometric Clustering

Case et al. (2006) showed that learnability in the limit of ▸ *clustering*, with or without additional information, depends strongly on geometric constraints on the shape of the clusters. In this approach the hypothesis space of possible clusters is pre-given in each setting. It was hoped to obtain thereby insight into the difficulty of clustering when the clusters are restricted to preassigned *geometrically* defined classes.

This is interestingly complementary to the *conceptual clustering* approach (see, e.g., Pitt and Reinke 1988; Mishra et al. 2004) where one restricts the possible clusters to have good "verbal" descriptions in some language.

Clustering of many of the geometric classes investigated was shown to *require* information *in addition* to a presentation, $d_0, d_1, d_2, \ldots$, of the set of points to be clustered. For example, for clusters as convex hulls of finitely many points in a rational vector space, clustering can be done – but with the number of clusters as additional information. Let $\mathcal{S}$ consist of all polygons including their interiors – in the rational two-dimensional plane *without intersections and degenerated angles*. (Attention was restricted to spaces of rationals since 1. computer reals are rationals, 2. this avoids the uncountability of the set of reals, and 3. this avoids dealing with *un*computable real points.) The class $\mathcal{S}$ *can* be clustered – but with the number of vertices of the polygons of the clusters involved as additional information.

Correspondingly, then, it was shown that the class $\mathcal{S}'$ containing $\mathcal{S}$ *together with* all such polygons but with one hole (the nondegenerate differences of two members in $\mathcal{S}$) can*not* be clustered with the number of vertices as additional information, yet $\mathcal{S}'$ can be clustered with *area* as additional information – and this even in higher dimensions and with any number of holes (Case et al. 2006).

It remains to be seen if some forms of geometrically constrained clustering can be usefully complementary to, say, conceptually/verbally constrained clustering.

Insights for Limitations of Science

We briefly treat below in some problems regarding parsimonious, refutable, and consistent hypotheses.

It is common wisdom in science that one should fit parsimonious explanations, hypotheses, or programs to data. In machine learning, this has been successfully applied, e.g., (Wallace and Dowe 1999; Wallace 2005).

Curiously, though, there are many results in inductive inference in which we see sometimes severe *degradations* of learning power caused by demanding *parsimonious* predictive programs (see, e.g., Freivalds 1975; Kinber 1977; Chen 1982; Case et al. 1996; Ambainis et al. 2004).

It is an interesting problem to resolve the seeming, likely not actual contradiction between the just prior two paragraphs.

Popper's Refutability (1962) asserts that hypotheses in science should be subject to refutation. Besides the well-known difficulties of Duhem–Quine (Harding 1976) of knowing which component hypothesis to throw out when a compound hypothesis badly fails to make correct predictions, inductive inference theorems have provided very different difficulties. Case and Smith (1983) outline cases of usefully *in*complete (hence wrong) hypothesis that cannot be refuted, and Case and Suraj (2007) (see also Case 2007) provide cases of inductively inferable higher order hypothesis not totally subject to refutation in cases where ordinary hypotheses subject to full refutation cannot be inductively inferred.

While Duhem–Quine may impact machine learning eventually, it remains to be seen about the inductive inference results of the just prior paragraph.

Requiring ▸ inductive inference procedures always to output an hypothesis in various senses *consistent* with (e.g., not ignoring) the data on which that hypothesis is based seems like mere common sense. However, from Bārzdiņš (1974a), Blum and Blum (1975), Wiehagen (1976), and Case et al. (2004), we see that strict adherence to various consistency principles can severely attenuate the learning power of inductive inference machines. Furthermore, interestingly, *even when inductive inference is polytime constrained*, we see similar counterintuitive results to the effect that a kind of consistency can strictly attenuate learning power (Wiehagen and Zeugmann 1994).

A machine learning analog might be Breiman's bagging (Breiman 1996) and random forests (Breiman 2001), where data is purposely ignored. However, in these cases, the purpose of ignoring data is to avoid overfitting to noise.

It remains to be seen, whether, in applied machine learning involving cases of practically noiseless data, one can also obtain some advantage in ignoring some consistency principles. Again the potential lesson from inductive inference is abstract and provides only a hint of something to work out in real machine learning problems.

Cross-References

▸ Behavioral Cloning
▸ Clustering
▸ Concept Drift
▸ Inductive Logic Programming

Recommended Reading

Ambainis A, Case J, Jain S, Suraj M (2004) Parsimony hierarchies for inductive inference. J Symb Logic 69:287–328

Angluin D, Gasarch W, Smith C (1989) Training sequences. Theor Comput Sci 66(3):255–272

Angluin D (1980) Finding patterns common to a set of strings. J Comput Syst Sci 21:46–62

Arikawa S, Shinohara T, Yamamoto A (1992) Learning elementary formal systems. Theor Comput Sci 95:97–113

Bain M, Sammut C (1999) A framework for behavioural cloning. In: Furakawa K, Muggleton S, Michie D (eds) Machine intelligence, vol 15. Oxford University Press, Oxford

Baluja S, Pomerleau D (1995) Using the representation in a neural network's hidden layer for task specific focus of attention. Technical report CMU-CS-95-143, School of Computer Science, CMU, May 1995. Appears in proceedings of the 1995 IJCAI

Bartlett P, Ben-David S, Kulkarni S (1996) Learning changing concepts by exploiting the structure of change. In: Proceedings of the ninth annual conference on computational learning theory, Desenzano del Garda. ACM Press, New York

Bartlmae K, Gutjahr S, Nakhaeizadeh G (1997) Incorporating prior knowledge about financial markets through neural multitask learning. In: Refenes APN, Burgess AN, Moody JE (eds) Decision technologies for computational finance. Proceedings of the fifth international conference on computational finance. Kluwer Academic, pp 425–432

Bārzdiņš J (1974a) Inductive inference of automata, functions and programs. In: Proceedings of the international congress of mathematicians, Vancouver, pp 771–776

Bārzdiņš J (1974b) Two theorems on the limiting synthesis of functions. In: Theory of algorithms and programs, vol 210. Latvian State University, Riga, pp 82–88

Blum L, Blum M (1975) Toward a mathematical theory of inductive inference. Inf Control 28:125–155

Blum A, Chalasani P (1992) Learning switching concepts. In: Proceedings of the fifth annual conference on computational learning theory, Pittsburgh. ACM Press, New York, pp 231–242

Bratko I, Muggleton S (1995) Applications of inductive logic programming. Commun ACM 38(11):65–70

Bratko I, Urbančič T, Sammut C (1998) Behavioural cloning of control skill. In: Michalski RS, Bratko I, Kubat M (eds) Machine learning and data mining: methods and applications. Wiley, New York, pp 335–351

Brazma A, Ukkonen E, Vilo J (1996) Discovering unbounded unions of regular pattern languages from positive examples. In: Proceedings of the seventh international symposium on algorithms and computation (ISAAC'96). Lecture notes in computer science, vol 1178. Springer, Berlin, pp 95–104

Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Caruana R (1993) Multitask connectionist learning. In: Proceedings of the 1993 connectionist models summer school. Lawrence Erlbaum, Hillsdale, pp 372–379

Caruana R (1996) Algorithms and applications for multitask learning. In: Proceedings 13th international conference on machine learning. Morgan Kaufmann, San Francisco, pp 87–95

Case J (1994) Infinitary self-reference in learning theory. J Exp Theor Artif Intell 6:3–16

Case J (1999) The power of vacillation in language learning. SIAM J Comput 28(6):1941–1969

Case J (2007) Directions for computability theory beyond pure mathematical. In: Gabbay D, Goncharov S, Zakharyaschev M (eds) Mathematical problems from applied logic II. New logics for the twenty-first century. International mathematical series, vol 5. Springer, New York

Case J, Kötzing T (2009) Difficulties in forcing fairness of polynomial time inductive inference. In: Gavalda R, Lugosi G, Zeugmann T, Zilles S (eds) 20th international conference on algorithmic learning theory (ALT'09). LNAI, vol 5809. Springer, Berlin, pp 263–277

Case J, Lynes C (1982) Machine inductive inference and language identification. In: Nielsen M, Schmidt E (eds) Proceedings of the 9th international colloquium on automata, languages and programming. Lecture notes in computer science, vol 140. Springer, Berlin, pp 107–115

Case J, Smith C (1983) Comparison of identification criteria for machine inductive inference. Theor Comput Sci 25:193–220

Case J, Suraj M (2007) Weakened refutability for machine learning of higher order definitions 2007. Working paper for eventual journal submission

Case J, Jain S, Kaufmann S, Sharma A, Stephan F (2001) Predictive learning models for concept drift (special issue for ALT'98). Theor Comput Sci 268:323–349

Case J, Jain S, Lange S, Zeugmann T (1999) Incremental concept learning for bounded data mining. Inf Comput 152:74–110

Case J, Jain S, Montagna F, Simi G, Sorbi A (2005) On learning to coordinate: random bits help, insightful normal forms, and competency isomorphisms (special issue for selected learning theory papers from COLT'03, FOCS'03, and STOC'03). J Comput Syst Sci 71(3):308–332

Case J, Jain S, Martin E, Sharma A, Stephan F (2006) Identifying clusters from positive data. SIAM J Comput 36(1):28–55

Case J, Jain S, Ott M, Sharma A, Stephan F (2000) Robust learning aided by context (special issue for COLT'98). J Comput Syst Sci 60:234–257

Case J, Jain S, Sharma A (1996) Machine induction without revolutionary changes in hypothesis size. Inf Comput 128:73–86

Case J, Jain S, Stephan F, Wiehagen R (2004) Robust learning – rich and poor. J Comput Syst Sci 69(2):123–165

Case J, Ott M, Sharma A, Stephan F (2002) Learning to win process-control games watching gamemasters. Inf Comput 174(1):1–19

Cenzer D, Remmel J (1992) Recursively presented games and strategies. Math Soc Sci 24:117–139

Chen K (1982) Tradeoffs in the inductive inference of nearly minimal size programs. Inf Control 52:68–86

de Garis H (1990a) Genetic programming: building nanobrains with genetically programmed neural network modules. In: IJCNN: international joint conference on neural networks, vol 3. IEEE Service Center, Piscataway, pp 511–516

de Garis H (1990b) Genetic programming: modular neural evolution for Darwin machines. In: Caudill M (ed) IJCNN-90-WASH DC; international joint conference on neural networks, vol 1. Lawrence Erlbaum Associates, Hillsdale, pp 194–197 de Garis H (1991) Genetic programming: building artificial nervous systems with genetically programmed neural network modules. In: Soušek B, The IRIS group (eds) Neural and intelligenct systems integeration: fifth and sixth generation integerated reasoning information systems, Chap. 8. Wiley, New York, pp 207–234

Devaney M, Ram A (1994) Dynamically adjusting concepts to accommodate changing contexts. In: Kubat M, Widmer G (eds) Proceedings of the ICML-96 pre-conference workshop on learning in context-sensitive domains, Bari. Journal submission

Dietterich T, Hild H, Bakiri G (1995) A comparison of ID3 and backpropogation for English text-tospeech mapping. Mach Learn 18(1):51–80

Fahlman S (1991) The recurrent cascade-correlation architecture. In: Lippmann R, Moody J, Touretzky D (eds) Advances in neural information processing systems, vol 3. Morgan Kaufmann Publishers, San Mateo, pp 190–196

Freivalds R (1975) Minimal Gödel numbers and their identification in the limit. Lecture notes in computer science, vol 32. Springer, Berlin, pp 219–225

Freund Y, Mansour Y (1997) Learning under persistent drift. In: Ben-David S, (ed) Proceedings of the third European conference on computational learning theory (EuroCOLT'97). Lecture notes in artificial intelligence, vol 1208. Springer, Berlin, pp 94–108

Fulk M (1990) Robust separations in inductive inference. In: Proceedings of the 31st annual symposium on foundations of computer science. IEEE Computer Society, St. Louis, pp 405–410

Harding S (ed) (1976) Can theories be refuted? Essays on the Duhem-Quine thesis. Kluwer Academic Publishers, Dordrecht

Helmbold D, Long P (1994) Tracking drifting concepts by minimizing disagreements. Mach Learn 14:27–46

Hildebrand F (1956) Introduction to numerical analysis. McGraw-Hill, New York

Jain S (1999) Robust behaviorally correct learning. Inf Comput 153(2):238–248

Jain S, Sharma A (1997) Elementary formal systems, intrinsic complexity, and procrastination. Inf Comput 132:65–84

Jain S, Sharma A (2002) Mind change complexity of learning logic programs. Theor Comput Sci 284(1):143–160

Jain S, Osherson D, Royer J, Sharma A (1999) Systems that learn: an introduction to learning theory, 2nd edn. MIT Press, Cambridge, MA

Jain S, Smith C, Wiehagen R (2001) Robust learning is rich. J Comput Syst Sci 62(1):178–212

Kilpeläinen P, Mannila H, Ukkonen E (1995) MDL learning of unions of simple pattern languages from positive examples. In: Vitányi P (ed) Computational learning theory, second European conference, EuroCOLT'95. Lecture notes in artificial intelligence, vol 904. Springer, Berlin, pp 252–260

Kinber E (1977) On a theory of inductive inference. Lecture notes in computer science, vol 56. Springer, Berlin, pp 435–440

Kinber E, Smith C, Velauthapillai M, Wiehagen R (1995) On learning multiple concepts in parallel. J Comput Syst Sci 50:41–52

Krishna Rao M (1996) A class of prolog programs inferable from positive data. In: Arikawa A, Sharma A (eds) Seventh international conference on algorithmic learning theory (ALT' 96). Lecture notes in artificial intelligence, vol 1160. Springer, Berlin, pp 272–284

Krishna Rao M (2000) Some classes of prolog programs inferable from positive data (Special Issue for ALT'96). Theor Comput Sci A 241:211–234

Krishna Rao M (2004) Inductive inference of term rewriting systems from positive data. In: Ben-David S, Case J, Maruoka A (eds) Algorithmic learning theory: fifteenth international conference (ALT'2004). Lecture notes in artificial intelligence, vol 3244. Springer, Berlin, pp 69–82

Krishna Rao M (2005) A class of prolog programs with non-linear outputs inferable from positive data. In: Jain S, Simon HU, Tomita E (eds) Algorithmic learning theory: sixteenth international conference (ALT'2005). Lecture notes in artificial intelligence, vol 3734. Springer, Berlin, pp 312–326

Krishna Rao M, Sattar A (1998) Learning from entailment of logic programs with local variables. In: Richter M, Smith C, Wiehagen R, Zeugmann T (eds) Ninth international conference on algorithmic learning theory (ALT'98). Lecture notes in artificial intelligence, vol 1501. Springer, Berlin, pp 143–157

Kubat M (1992) A machine learning based approach to load balancing in computer networks. Cybern Syst 23:389–400

Kummer M, Ott M (1996) Learning branches and learning to win closed recursive games. In: Proceedings of the ninth annual conference on computational learning theory, Desenzano del Garda. ACM Press, New York

Lange S, Wiehagen R (1991) Polynomial time inference of arbitrary pattern languages. New Gener Comput 8:361–370

Lavrač N, Džeroski S (1994) Inductive logic programming: techniques and applications. Ellis Horwood, New York

Maler O, Pnueli A, Sifakis J (1995) On the synthesis of discrete controllers for timed systems. In: Proceedings of the annual symposium on the theoretical aspects of computer science. LNCS, vol 900. Springer, Berlin, pp 229–242

Matwin S, Kubat M (1996) The role of context in concept learning. In: Kubat M, Widmer G (eds) Proceedings of the ICML-96 pre-conference workshop on learning in context-sensitive domains, Bari, pp 1–5

Maye A, Hsieh C, Sugihara G, Brembs B (2007) Order in spontaneous behavior. PLoS One, May 2007. http://brembs.net/spontaneous/

Mishra N, Ron D, Swaminathan R (2004) A new conceptual clustering framework. Mach Learn 56 (1–3):115–151

Mitchell T (1997) Machine learning. McGraw Hill, New York

Mitchell T, Caruana R, Freitag D, McDermott J, Zabowski D (1994) Experience with a learning, personal assistant. Commun ACM 37:80–91

Montagna F, Osherson D (1999) Learning to coordinate: a recursion theoretic perspective. Synthese 118:363–382

Muggleton S, De Raedt L (1994) Inductive logic programming: theory and methods. J Logic Program 19/20:669–679

Odifreddi P (1999) Classical recursion theory, vol II. Elsivier, Amsterdam

Osherson D, Stob M, Weinstein S (1986) Systems that learn: an introduction to learning theory for cognitive and computer scientists. MIT Press, Cambridge, MA

Ott M, Stephan F (2002) Avoiding coding tricks by hyperrobust learning. Theor Comput Sci 284(1):161–180

Pitt L, Reinke R (1988) Criteria for polynomial-time (conceptual) clustering. Mach Learn 2:371–396

Popper K (1992) Conjectures and refutations: the growth of scientific knowledge. Basic Books, New York

Pratt L, Mostow J, Kamm C (1991) Direct transfer of learned information among neural networks. In: Proceedings of the 9th national conference on artificial intelligence (AAAI-91), Anaheim. AAAI press, Menlo Park

Rogers H (1987) Theory of recursive functions and effective computability. McGraw Hill, New York. (Reprinted, MIT Press, 1987)

Salomaa A (1994a) Patterns (The formal language theory column). EATCS Bull 54:46–62

Salomaa A (1994b) Return to patterns (The formal language theory column). EATCS Bull 55: 144–157

Sejnowski T, Rosenberg C (1986) NETtalk: a parallel network that learns to read aloud. Technical report JHU-EECS-86-01, Johns Hopkins University

Shimozono S, Shinohara A, Shinohara T, Miyano S, Kuhara S, Arikawa S (1994) Knowledge acquisition from amino acid sequences by machine learning system BONSAI. Trans Inf Process Soc Jpn 35:2009–2018

Shinohara T (1983) Inferring unions of two pattern languages. Bull Inf Cybern 20:83–88

Shinohara T, Arikawa A (1995) Pattern inference. In: Jantke KP, Lange S (eds) Algorithmic learning for knowledge-based systems. Lecture notes in artificial intelligence, vol 961. Springer, Berlin, pp 259–291

Smullyan R (1961) Theory of formal systems. Annals of mathematics studies, vol 47). Princeton University Press, Princeton

Šuc D (2003) Machine reconstruction of human control strategies. Frontiers in artificial intelligence and applications, vol 99. IOS Press, Amsterdam

Thomas W (1995) On the synthesis of strategies in infinite games. In: Proceedings of the annual symposium on the theoretical aspects of computer science. LNCS, vol 900. Springer, Berlin, pp 1–13

Thrun S (1996) Is learning the n-th thing any easier than learning the first? In: Advances in neural information processing systems, vol 8. Morgan Kaufmann, San Mateo

Thrun S, Sullivan J (1996) Discovering structure in multiple learning tasks: the TC algorithm. In: Proceedings of the thirteenth international conference on machine learning (ICML-96). Morgan Kaufmann, San Francisco, pp 489–497

Tsung F, Cottrell G (1989) A sequential adder using recurrent networks. In: IJCNN-89-WASHINGTON DC: international joint conference on neural networks, 18–22 June, vol 2. IEEE Service Center, Piscataway, pp 133–139

Waibel A (1989a) Connectionist glue: modular design of neural speech systems. In: Touretzky D, Hinton G, Sejnowski T (eds) Proceedings of the 1988 connectionist models summer school. Morgan Kaufmann, San Mateo, pp 417–425

Waibel A (1989b) Consonant recognition by modular construction of large phonemic time-delay neural networks. In: Touretzky DS (ed) Advances in neural information processing systems I. Morgan Kaufmann, San Mateo, pp 215–223

Wallace C (2005) Statistical and inductive inference by minimum message length. Information science and statistics. Springer, New York. Posthumously published

Wallace C, Dowe D (1999) Minimum message length and Kolmogorov complexity (special issue on Kolmogorov complexity). Comput J 42(4):123–155. http://comjnl.oxfordjournals.org/cgi/reprint/42/4/270

Widmer G, Kubat M (1996) Learning in the presence of concept drift and hidden contexts. Mach Learn 23:69–101

Wiehagen R (1976) Limes-Erkennung rekursiver Funktionen durch spezielle Strategien. Electronische Informationverarbeitung und Kybernetik 12: 93–99

Wiehagen R, Zeugmann T (1994) Ignoring data may be the only way to learn efficiently. J Exp Theor Artif Intell 6:131–144

Wright K (1989) Identification of unions of languages drawn from an identifiable class. In: Rivest R, Haussler D, Warmuth M (eds) Proceedings of the second annual workshop on computational learning theory, Santa Cruz. Morgan Kaufmann Publishers, San Mateo, pp 328–333

Wrobel S (1994) Concept formation and knowledge revision. Kluwer Academic Publishers, Dordrecht

Zeugmann T (1986) On Bārzdiņš' conjecture. In: Jantke KP (ed) Proceedings of the international workshop on analogical and inductive inference. Lecture notes in computer science, vol 265. Springer, Berlin, pp 220–227

Zeugmann T (1998) Lange and Wiehagen's pattern language learning algorithm: an average case analysis with respect to its total learning time. Ann Math Artif Intell 23:117–145

Connectivity

▸ Topology of a Neural Network

Consensus Clustering

Synonyms

Clustering aggregation; Clustering ensembles

Definition

In Consensus Clustering we are given a set of n objects V, and a set of m clusterings $\{C_1, C_2, \ldots, C_m\}$ of the objects in V. The aim is to find a single clustering C that *disagrees* least with the input clusterings, that is, C minimizes

$$D(C) = \sum_{C_i} d(C, C_j),$$

for some metric d on clusterings of V. Meilă (2003) proposed the principled *variation of information* metric on clusterings, but it has been difficult to analyze theoretically. The Mirkin metric is the most widely used, in which $d(C, C')$ is the number of pairs of objects (u, v) that are clustered together in C and apart in C', or vice versa; it can be calculated in time $O(mn)$.

We can interpret each of the clusterings C_i in Consensus Clustering as evidence that pairs ought be put together or separated. That is, w^i_{uv} is the number of C_i in which $C_i[u] = C_i[v]$ and w^-_{uv} is the number of C_i in which $C_i[u] \neq C_i[v]$. It is clear that $w^+_{uv} + w^-_{uv} = m$ and that Consensus clustering is an instance of Correlation clustering in which the w^-_{uv} weights obey the triangle inequality.

Constrained Clustering

Kiri L. Wagstaff
Pasadena, CA, USA

Definition

Constrained clustering is a semisupervised approach to ▸ clustering data while incorporating domain knowledge in the form of constraints. The constraints are usually expressed as pairwise statements indicating that two items must, or cannot, be placed into the same cluster. Constrained clustering algorithms may enforce every constraint in the solution, or they may use the constraints as guidance rather than hard requirements.

Motivation and Background

▸ Unsupervised learning operates without any domain-specific guidance or preexisting knowledge. Supervised learning requires that all training examples be associated with labels. Yet it is often the case that existing knowledge for a problem domain fits neither of these extremes. Semisupervised learning methods fill this gap by making use of both labeled and unlabeled data. Constrained clustering, a form of semisupervised learning, was developed to extend clustering algorithms to incorporate existing domain knowledge, when available. This knowledge may arise from labeled data or from more general rules about the concept to be learned.

One of the original motivating applications was noun phrase coreference resolution, in which noun phrases in a text must be clustered together to represent distinct entities (e.g., "Mr. Obama" and "the President" and "he", separate from "Sarah Palin" and "she" and "the Alaska governor"). This problem domain contains several natural rules for when noun phrases should (such as appositive phrases) or should not (such as a mismatch on gender) be clustered together. These rules can be translated into a collection of pairwise constraints on the data to be clustered.

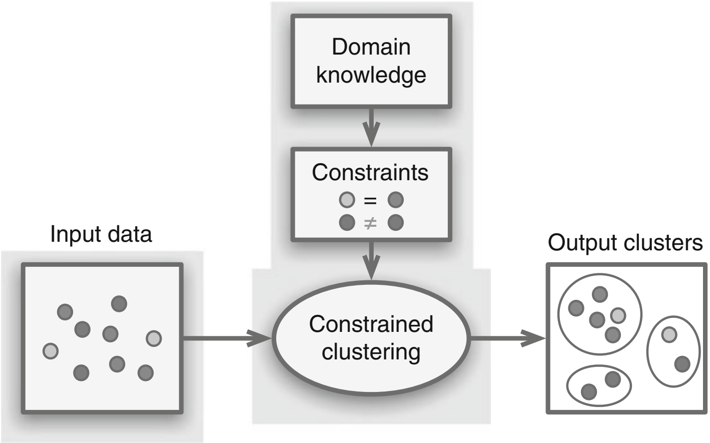
Constrained clustering algorithms have now been applied to a rich variety of domain areas, including hyperspectral image analysis, road lane divisions from GPS data, gene expression microarray analysis, video object identification, document clustering, and web search result grouping.

Structure of the Learning System

Constrained clustering arises out of existing work with unsupervised clustering algorithms. In this description, we focus on clustering algorithms that seek a partition of the data into disjoint clusters, using a distance or similarity measure to place similar items into the same cluster. Usually, the desired number of clusters, k, is specified as an input to the algorithm. The most common clustering algorithms are k-means (MacQueen 1967) and expectation maximization or EM (Dempster et al. 1977) (Fig. 1).

A constrained clustering algorithm takes the same inputs as a regular (unsupervised) clustering algorithm and also accepts a set of pairwise constraints. Each constraint is a must-link or ▸ cannot-link constraint. The must-link constraints form an equivalence relation, which permits the inference of additional transitively implied must-links as well as additional entailed cannot-link constraints between items from distinct must-link cliques. Specifying a significant number of pairwise constraints might be tedious for large data sets, so often they may be generated from a manually labeled subset of the data or from domain-specific rules.

The algorithm may interpret the constraints as hard constraints that must be satisfied in the output or as soft preferences that can be violated, if necessary. The former approach was used in the first constrained clustering algorithms, COP-COBWEB (Wagstaff and Cardie 2000) and COP-kmeans (Wagstaff et al. 2001). COP-kmeans accommodates the constraints by restricting item assignments to exclude any constraint violations. If a solution that satisfies the constraints is not found, COP-kmeans terminates without a solution. Later, algorithms such as PCK-means and MPCK-means (Bilenko et al. 2004) permitted the violation of constraints when necessary by introducing a violation penalty. This is useful when the constraints may contain noise or internal inconsistencies, which are especially relevant in real-world domains. Constrained versions of other clustering algorithms such as EM (Shental et al. 2004) and spectral clustering (Kamvar et al. 2003) also exist. Penalized probabilistic clustering (PPC) is a modified version of EM that interprets the constraints as (soft) probabilistic priors on the relationships between items (Lu and Leen 2005).



Constrained Clustering, Fig. 1 The constrained clustering algorithm takes in nine items and two pairwise constraints (one must-link and one cannot-link). The output clusters respect the specified constraints

In addition to constraining the assignment of individual items, constraints can be used to learn a better distance metric for the problem at hand (Bar-Hillel et al. 2005; Klein et al. 2002; Xing et al. 2003). Must-link constraints hint that the effective distance between those items should be low, while cannot-link constraints suggest that their pairwise distance should be high. Modifying the metric accordingly permits the subsequent application of a regular clustering algorithm, which need not explicitly work with the constraints at all. The MPCK-means algorithm fuses these approaches together, providing both constraint satisfaction and metric learning simultaneously (Basu et al. 2004; Bilenko et al. 2004).

More information about subsequent advances in constrained clustering algorithms, theory, and novel applications can be found in a compilation edited by Basu et al. (2008).

Programs and Data

The MPCK-means algorithm is available in a modified version of the Weka machine learning toolkit (Java) at http://www.cs.utexas.edu/users/ml/risc/code/.

Recommended Reading

Bar-Hillel A, Hertz T, Shental N, Weinshall D (2005) Learning a Mahalanobis metric from equivalence constraints. J Mach Learn Res 6:937–965

Basu S, Bilenko M, Mooney RJ (2004) A probabilistic framework for semi-supervised clustering. In: Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining, Seattle, pp 59–68

Basu S, Davidson I, Wagstaff K (eds) (2008) Constrained clustering: advances in algorithms, theory, and applications. CRC Press, Boca Raton

Bilenko M, Basu S, Mooney RJ (2004) Integrating constraints and metric learning in semi-supervised clustering. In: Proceedings of the twenty-first international conference on machine learning, Banff, pp 11–18

Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J R Stat Soc 39(1):1–38

Kamvar S, Klein D, Manning CD (2003) Spectral learning. In: Proceedings of the international joint conference on artificial intelligence, Acapulco, pp 561–566

Klein D, Kamvar SD, Manning CD (2002) From instance-level constraints to space-level constraints: making the most of prior knowledge in data clustering. In: Proceedings of the nineteenth international conference on machine learning, Sydney, pp 307–313

Lu Z, Leen T (2005) Semi-supervised learning with penalized probabilistic clustering. In: Advances in neural information processing systems, vol 17. MIT Press, Cambridge, MA, pp 849–856

MacQueen JB (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth symposium on math, statistics, and probability, vol 1. University of California Press, California, pp 281–297

Shental N, Bar-Hillel A, Hertz T, Weinshall D (2004) Computing Gaussian mixture models with EM using equivalence constraints. In: Advances in neural information processing systems, vol 16. MIT Press, Cambridge, MA, pp 465–472

Wagstaff K, Cardie C (2000) Clustering with instance-level constraints. In: Proceedings of the seventeenth international conference on machine learning. Morgan Kaufmann, San Francisco, pp 1103–1110

Wagstaff K, Cardie C, Rogers S, Schroedl S (2001) Constrained k-means clustering with background knowledge. In: Proceedings of the eighteenth international conference on machine learning. Morgan Kaufmann, San Francisco, pp 577–584

Xing EP, Ng AY, Jordan MI, Russell S (2003) Distance metric learning, with application to clustering with side-information. In: Advances in neural information processing systems, vol 15. MIT Press, Cambridge, MA, pp 505–512

Constraint Classification

▶ Preference Learning

Constraint-Based Mining

Siegfried Nijssen
Katholieke Universiteit Leuven, Leuven,
Belgium

Definition

Constraint-based mining is the research area studying the development of data mining algorithms that search through a pattern or model space restricted by constraints. The term is usually used to refer to algorithms that search for patterns only. The most well-known instance of constraint-based mining is the mining of ▸ frequent patterns. Constraints are needed in pattern mining algorithms to increase the efficiency of the search and to reduce the number of patterns that are presented to the user, thus making knowledge discovery more effective and useful.

Motivation and Background

Constraint-based pattern mining is a generalization of frequent itemset mining. For an introduction to frequent itemset mining, see ▸ Frequent Pattern. A constraint-based mining problem is specified by providing the following elements:

- A database $\mathcal{D}$, usually consisting of independent transactions (or instances)
- A ▸ hypothesis space $\mathcal{L}$ of patterns
- A constraint $q(\theta, \mathcal{D})$ expressing criteria that a pattern θ in the hypothesis space should fulfill on the database

The general constraint-based mining problem is to find the set

$$\mathrm{Th}(\mathcal{D}, \mathcal{L}, q) = \{\theta \in \mathcal{L} | q(\theta, \mathcal{D}) = \mathrm{true}\}.$$

Alternative problem settings are obtained by making different choices for $\mathcal{D}, \mathcal{L}$ and q. For instance,

- If the database and hypothesis space consist of itemsets, and the constraint checks if the support of a pattern exceeds a predefined threshold in data, the frequent itemset mining problem is obtained (see ▸ Frequent Pattern)
- If the database and the hypothesis space consist of graphs or trees instead of itemsets, a graph mining or a tree mining problem is obtained. For more information about these topics, see ▸ Graph Mining and ▸ Tree Mining
- Additional syntactic constraints can be imposed

An overview of important types of constraints is given below.

One can generalize the constraint-based mining problem beyond pattern mining. Also models, such as ▸ Decision Trees, could be seen as languages of interest. In the broadest sense, topics such as ▸ Constrained Clustering, ▸ Cost-Sensitive Learning, and even learning ▸ Support Vector Machines (SVMs) may be seen as constraint-based mining problems. However, it is currently not common to categorize these topics as *constraint-based mining*; in practice, the term refers to constraint-based *pattern* mining.

From the perspective of constraint-based mining, the knowledge discovery process can be seen as a process in which a user repeatedly specifies constraints for data mining algorithms; the data mining system is a solver that finds patterns or models that satisfy the constraints.

This approach to data mining is very similar to querying relational databases. Whereas relational databases are usually queried using operations such as projections, selections, and joins, in the constraint-based mining framework data is queried to find patterns or models that satisfy constraints that cannot be expressed in these primitives. A database which supports constraint-based mining queries, stores patterns and models, and allows later reuse of patterns and models, is sometimes also called an *inductive database* (Imielinski and Mannila 1996).

Structure of the Learning System

Constraints

Frequent pattern mining algorithms can be generalized along several dimensions.

One way to generalize pattern mining algorithms is to allow them to deal with arbitrary coverage relations, which determine when a pattern matches a transaction in the data. In the example of mining itemsets, the subset relation determines the coverage relation. The coverage relation is at the basis of constraints such as minimum support; an alternative coverage relation would be the superset relation.

From the coverage relation follows a generality relationship. A pattern θ_1 is defined to be more specific than a pattern θ_2 (denoted by $\theta_1 \succ \theta_2$) if any transaction that is covered by θ_1 is also covered by θ_2 (see ▸ Generalization). In frequent itemset mining, itemset I_1 is more general than itemset I_2 if and only $I_1 \subseteq I_2$.

Generalization and coverage relationships can be used to identify the following types of constraints.

Monotonic and Anti-Monotonic Constraints

An essential property which is exploited in ▸ frequent pattern mining, is that allsubsets of a frequent pattern are also frequent. This is a property that can begeneralized:

- A constraint is called *monotonic* if any generalization of a pattern that satisfies the constraint, also satisfies the constraint
- A constraint is called *anti-monotonic* if any specialization of a pattern that satisfies the constraint, also satisfies the constraint

In some publications, the definitions of monotonic and anti-monotonic are used reversely.

The following are examples of monotonic constraints:

- Minimum support
- Syntactic constraints, for instance: a constraint that requires that patterns specializing a given pattern x are excluded a constraint requiring patterns to be small given a definition of pattern size
- Disjunctions or conjunctions of monotonic constraints
- Negations of anti-monotonic constraints

The following are examples of anti-monotonic constraints:

- Maximum support
- Syntactic constraints, for instance, a constraint that requires that patterns generalizing a given pattern x are excluded
- Disjunctions or conjunctions of anti-monotonic constraints
- Negations of monotonic constraints

Succinct Constraints

Constraints that can be pushed in the mining process by adapting the pattern space or data, are called succinct constraints. An example of a succinct constraint is the monotonic constraint that an itemset should contain the item A. This constraint could be dealt with by deleting all transactions that do not contain A. For any frequent itemset found in the new dataset, it is now known that the item A can be added to it.

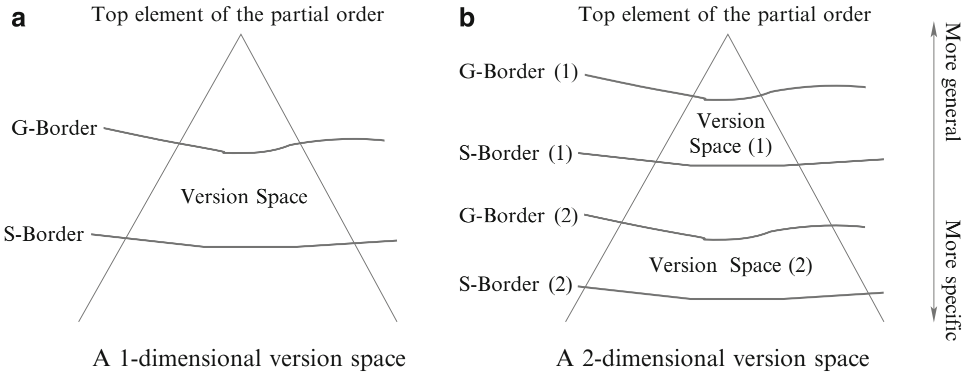
Convertible Constraints

Some constraints that are not monotonic, can still be *convertible* monotonic (Pei and Han 2002). A constraint is convertible monotonic if for every pattern θ one least general generalization θ' can be identified such that if θ satisfies the constraint, then θ' also satisfies the constraint. An example of a convertible constraint is a maximum average cost constraint. Assume that every item in an itemset has a cost as defined by a function $c(i)$. The constraint $c(I) = \sum_{i \in I} c(i)/|I| \leq maxcost$ is not monotonic. However, for every itemset I with $c(I) \leq maxcost$, if an item i is removed with $c(i) = \max_{i \in I} c(i)$, an itemset with $c(I - \{i\}) \leq c(I) \leq maxcost$ is obtained.

Maximum average cost has the desirable property that no access to the data is needed to identify the generalization that should satisfy the constraints. If it is not possible to identify the necessary least general generalization before accessing the data, the convertible constraint is also sometimes called weak (anti-)monotone (Zhu et al. 2007).

Boundable Constraints

Constraints on non-monotonic measures for which a monotonic bound exist, are called boundable. An example of such a constraint is a minimum accuracy constraint in a database with binary class labels. Assume that every itemset is interpreted as a rule **if** I **then** 1 **else** 2 (thus, class label 1 is predicted if a transaction contains itemset I, or class label 2 otherwise; see ▸ Supervised Descriptive Rule Induction). A minimum accuracy constraint can be formalized by the formula $(\mathrm{fr}(I, D_1) + |D_2| - \mathrm{fr}(I, D_2))/|D| \geq minacc$, where D_k is the



Constraint-Based Mining, Fig. 1 Version spaces

database containing only the examples labeled with class label k. It can be derived from this that

$$\mathrm{fr}(I, D_1) \geq |D| minacc - |D_2| +$$
$$\mathrm{fr}(I, D_2) \geq |D| minacc - |D_2|.$$

In other words, if a high accuracy is desirable, a minimum number of examples of class 1 is required to be covered, and a minimum frequency constraint can thus be derived. Therefore, minimum support can be used as a bound for minimum accuracy.

The principle of deriving bounds for non-monotonic measures can be applied widely (Bayardo et al. 1999; Morishita and Sese 2000).

Borders

If constraints are not restrictive enough, the number of patterns can be huge. Ignoring statistics about patterns such as their exact frequency, the set of patterns can be represented more compactly only by listing the patterns in the *border*(s) (Mannila and Toivonen 1997), similar to the idea of ▶ version spaces. An example of a border is the set of maximalfrequent itemsets (see ▶ Frequent Pattern). Borders can be computed for othertypes of both monotonic and anti-monotonic constraints as well. There areseveral complications compared to the simple frequent pattern miningsetting:

- If there is an anti-monotonic constraint, such as maximum support, not only is it needed to compute a border for the most specific elements in the set (S-Set), but also a border for the least general elements in the set (G-Set)
- If the formula is a disjunction of conjunctions, the result of a query becomes a union of version spaces, which is called a multi-dimensional version space (see Fig. 1) (De Raedt et al. 2002); the G-Set of one version space may be more general than the G-Set of another version space

Both the S-Set and the G-Set can be represented by listing elements just within the version space (the positive border), or elements just outside the version space (the negative border). For instance, the positive border of the G-Set consists of those patterns which are part of the version space, and for which no generalizations exist which are part of the version space.

Similarly, there may exist several representations of multi-dimensional version spaces; optimizing the representation of multi-dimensional version spaces is analogous to optimizing queries in relational databases (De Raedt et al. 2002).

Borders form a *condensed representations*, that is, they compactly represent the solution space; see ▶ Frequent Pattern.

Algorithms

For many of the constraints specified in the previous section specialized algorithms have been developed in combination with specific hypothesis spaces. It is beyond the scope of this chapter to discuss all these algorithms; only the most common ideas are provided here.

The main idea is that ▸ Apriori can easily be updated to deal with general monotonic constraints in arbitrary hypothesis spaces. The concept of a specialization refinement operator is essential to operateon other hypothesis spaces than itemsets. A specialization operator $\rho(\theta)$ computes a set of specializations in the hypothesis space for a given input pattern. In pattern mining, this operator should have the following properties:

- Completeness: every pattern in the hypothesis space should be reachable by repeated application of the refinement operator starting from the most general pattern in the hypothesis space
- Nonredundancy: every pattern in the hypothesis space should be reachable in only one way starting from the most general pattern in the hypothesis space

In itemset mining, optimal refinement is usually obtained by first ordering the items (for instance, alphabetically, or by frequency), and then adding items that are higher in the chosen order to a set than the items already in the set. For instance, for the itemset $\{A, C\}$, the specialization operator returns $\rho(\{A, C\}) = \{\{A, C, D\}, \{A, C, E\}\}$, assuming that the domain of items $\{A, B, C, D, E\}$ is considered. Other refinement operators are needed while dealing with other hypothesis spaces, such as in ▸ graph mining.

The search in Apriori proceeds breadth-first. Each level, the specialization operator is applied on patterns satisfying the monotonic constraints to generate candidates for the next level. For every new candidate it is checked whether its generalizations satisfy the monotonic constraints. To create a set of generalizations, a generalization refinement operator can be used. In frequent itemset mining, usually single items are removed from the itemset to generate generalizations.

More changes are required to deal with *anti-monotonic* constraints. A simple way of dealing with both monotonic and anti-monotonic constraints is to first compute all patterns that satisfy the monotonic constraints, and then to prune the patterns that fail to satisfy the anti-monotonic constraints. More challenging is to "push" anti-monotonic constraints in the mining process. An observation which is often exploited is that generalizations of patterns that do not satisfy the anti-monotonic constraint need not be considered. Well-known strategies are:

- In a breadth-first setting: traverse the lattice in reverse order for monotonic constraints, after the patterns have been determined satisfying the anti-monotonic constraints (De Raedt et al. 2002)
- In a depth-first setting: during the search for patterns, try to guess the largest pattern that can still be reached, and prune a branch in the search if the pattern does not satisfy the monotonic constraint on this pattern (Bucila et al. 2003; Kifer et al. 2003)

It is beyond the scope of this chapter to discuss how to deal with other types of constraints; however, it should be pointed out that not all combinations of constraints and hypothesis spaces have been studied; it is not obvious whether all constraints can be pushed usefully in a pattern search for any hypothesis space, for instance, when boundable constraints in more complex hypothesis spaces (such as graphs) are involved. Research in this area is ongoing.

Cross-References

▸ Constrained Clustering
▸ Frequent Pattern
▸ Graph Mining
▸ Tree Mining

Recommended Reading

Bayardo RJ Jr, Agrawal R, Gunopulos D (1999) Constraint-based rule mining in large, dense databases. In: Proceedings of the 15th international conference on data engineering (ICDE), Sydney, pp 188–197

Bucila C, Gehrke J, Kifer D, White WM (2003) DualMiner: a dual-pruning algorithm for itemsets with constraints. Data Min Knowl Discov 7(3):241–272

De Raedt L, Jaeger M, Lee SD, Mannila H (2002) A theory of inductive query answering (extended abstract). In: Proceedings of the second IEEE international conference on data mining (ICDM). IEEE Press, Los Alamitos, pp 123–130

Imielinski T, Mannila H (1996) A database perspective on knowledge discovery. Commun ACM 39:58–64

Kifer D, Gehrke J, Bucila C, White WM (2003) How to quickly find a witness. In: Proceedings of the twenty-second ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems. ACM Press, San Diego, pp 272–283

Mannila H, Toivonen H (1997) Levelwise search and borders of theories in knowledge discovery. Data Min Knowl Discov 1(3):241–258

Morishita S, Sese J (2000) Traversing itemset lattices with statistical metric pruning. In: Proceedings of the nineteenth ACM SIGACT-SIGMOD-SIGART symposium on database systems (PODS). ACM Press, San Diego, pp 226–236

Pei J, Han J (2002) Constrained frequent pattern mining: a pattern-growth view. SIGKDD Explor 4(1):31–39

Zhu F, Yan X, Han J, Yu PS (2007) gPrune: a constraint pushing framework for graph pattern mining. In: Proceedings of the sixth Pacific-Asia conference on knowledge discovery and data mining (PAKDD). Lecture notes in computer science, vol 4426. Springer, Berlin, pp 388–400

Constructive Induction

Constructive induction is any form of ▶ induction that generates new descriptors not present in the input data (Dietterich and Michalski 1983).

Recommended Reading

Dietterich TG, Michalski RS (1983) A comparative review of selected methods for learning from examples. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning: an artificial intelligence approach, Tioga, pp 41–81

Content Match

▶ Text Mining for Advertising

Content-Based Filtering

Synonyms

Content-based recommending

Definition

Content-based filtering is prevalent in ▶ Information Retrieval, where the text and multimedia content of documents is used to select documents relevant to a user's query. In the context this refers to content-based recommenders, that provide recommendations by comparing representations of content describing an item to representations of content that interests a user.

Content-Based Recommending

▶ Content-Based Filtering

Context-Sensitive Learning

▶ Concept Drift

Contextual Advertising

▶ Text Mining for Advertising

Continual Learning

Synonyms

Life-long learning

Definition

A learning system that can continue adding new data without the need to ever stop or freeze the updating. Usually continual learning requires incremental and ▶ online learning as a component, but not every incremental learning system has the ability to achieve continual learning, i.e., the learning may deterioate after some time.

Cross-References

▶ Cumulative Learning

Continuous Attribute

A **continuous attribute** can assume all values on the number line within the value range. See ▶ Attribute and ▶ Measurement Scales.

Contrast Set Mining

Definition

Contrast set mining is an area of ▶ supervised descriptive rule induction. The contrast set mining problem is defined as finding contrast sets, which are conjunctions of attributes and values that differ meaningfully in their distributions across groups (Bay and Pazzani 2001). In this context, groups are the properties of interest.

Recommended Reading

Bay SD, Pazzani MJ (2001) Detecting group differences: mining contrast sets. Data Mining Knowl Discov 5(3):213–246

Cooperative Coevolution

▶ Compositional Coevolution

Co-reference Resolution

▶ Entity Resolution
▶ Record Linkage

Correlation Clustering

Anthony Wirth
The University of Melbourne, Melbourne, VLC, Australia

Synonyms

Clustering with advice; Clustering with constraints; Clustering with qualitative information; Clustering with side information

Definition

In its rawest form, *correlation clustering* is graph optimization problem. Consider a ▶ clustering C to be a mapping from the elements to be clustered, V , to the set $\{1, \ldots, |V|\}$, so that u and v are in the same cluster if and only if $C[u] = C[v]$. *Given* a collection of items in which each pair (u, v) has two weights w^+_{uv} and w^-_{uv}, we must *find* a clustering C that minimizes

$$\sum_{C[u]=C[v]} w^-_{uv} + \sum_{C[u]\neq C[v]} w^+_{uv}, \qquad (1)$$

or, equivalently, maximizes

$$\sum_{C[u]=C[v]} w^+_{uv} + \sum_{C[u]\neq C[v]} w^-_{uv}. \qquad (2)$$

Note that although w^+_{uv} and w^-_{uv} may be thought of as positive and negative evidence towards coassociation, the actual weights are nonnegative.

Motivation and Background

The notion of *clustering with advice*, that is nonmetric-driven relations between items, had been studied in other communities (Ferligoj and Batagelj 1982) prior to its appearance in theoretical computer science. Traditional clustering problems, such as k-median and k-center, assume that there is some type of distance measure (metric) on the data items, and often specify the number of clusters that should be formed. In the clustering with advice framework, however, the number of clusters to be built need not be specified in advance: it can be an outcome of the objective function. Furthermore, instead of, or in addition to, a distance function, we are given advice as to which pairs of items are similar. The two weights w^+_{uv} and w^-_{uv} correspond to external advice about whether the pair should be clustered together or separately. Bansal et al. (2002) introduced the problem to the theoretical computer science and machine-learning communities. They were motivated by database consistency problems, in which the same entity appeared in different forms in various databases. Given a collection of such records from multiple databases, the aim is to cluster together the records that appear to correspond to the same entity. From this viewpoint, the log odds ratio from some classifier,

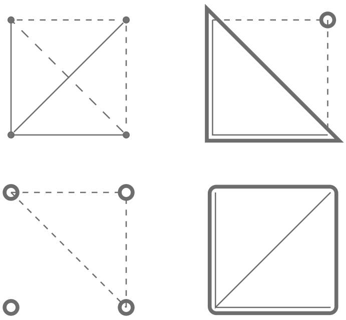
$$\log\left(\frac{\Pr(\text{same})}{\Pr(\text{different})}\right),$$

corresponds to a label w_{uv} for the pair. In many applications only one of the $+$ and $-$ weights for the pair is nonzero, that is

$$(w^+_{uv}, w^-_{uv}) = \begin{cases} (w_{uv}, 0) & \text{for } w_{uv} \geq 0 \\ (0, -w_{uv}) & \text{for } w_{uv} \leq 0. \end{cases}$$

In addition, if every pair has weight ±1, then the instance is called *complete*, otherwise it is referred to as *general*. Demaine et al. (2006) suggest the following motivation. Suppose we have a set of guests at a party. Each guest has preferences for whom they would like to sit with, and for whom they would like to avoid. We must group the guests into tables in a way that enhances the amicability of the party.

The notion of producing good clusterings when given inconsistent advice first appeared in the work of Ben et al. (1999). A canonical example of inconsistent advice is this: items u and v are similar, items v and y are similar, but u and y are dissimilar. It is impossible to find a clustering that satisfies all the advice. Figure 1 shows a very simple example of inconsistent advice. In addition, although Correlation clustering is an NP-hard problem, recent algorithms for clustering with advice *guarantee* that their solutions are only a specified factor worse than the optimal: that is, they are *approximation algorithms*.



Correlation Clustering, Fig. 1 *Top left* is a toy *clustering with advice* example showing three similar pairs (*solid edges*) and three dissimilar pairs (*dashed edges*). *Bottom left* is a clustering solution for this example with four singleton clusters, while *bottom right* has one cluster. *Top right* is a partitioning into two clusters that appears to best respect the advice

Theory

In setting out the correlation clustering framework, Bansal et al. (2002) noted that the following algorithm produces a 2-approximation for the maximization problem:

> If the total of the positive weights exceeds the total of the negative weights then, place all the items in a single cluster; otherwise, make each item a singleton cluster.

They then showed that complete instances are NP-hard to optimize, and how to minimize the penalty (1) with a constant factor approximation. The constant for this combinatorial algorithm was rather large. The algorithm relied heavily on the completeness of the instance; it iteratively *cleans* clusters until every cluster is *δ-clean*. That is, for each item at most a fraction $\delta (0 < \delta < 1)$ of the other items in its cluster have a negative relation with it, and at most δ outside its cluster a positive relation. Bansal et al. also demonstrated that the minimization problem on general instances is APX-hard: there is some constant, larger than 1, below which approximation is NP-hard. Finally, they provided a polynomial time approximation scheme (PTAS) for maximizing (2) in complete instances.

The constant factor for minimizing (1) on complete instances was improved to 4 by Charikar et al. (2003). They employed a region-growing type procedure to round the solution of a linear programming relaxation of the problem:

$$\begin{aligned}&\text{maximize}\\&\quad\sum_{ij} w_{ij}^{+}\cdot x_{ij} + w_{ij}^{-}\cdot(1-x_{ij})\\&\text{subject to}\\&\quad x_{ik} \le x_{ij} + x_{jk} \quad \text{for all } i,j,k\\&\quad x_{ij} \in [0,1] \qquad\quad \text{for all } i,j\end{aligned} \tag{3}$$

In this setting, $x_{ij} = 1$ implies i and j's separation, while $x_{ij} = 0$ implies coclustering, with values in between representing partial evidence. In practice solving this linear program is very slow and has huge memory demands (Bertolacci and Wirth 2007). Charikar et al. also showed that this version of problem is APX-hard.

For the maximization problem (2), they showed that instances with general weights were APX-hard and provided a rounding of the following semidefinite program (SDP) that yields a 0.7664 factor approximation algorithm.

$$\begin{aligned}&\text{maximize}\\&\quad\sum_{+(ij)} w_{ij}(v_i\cdot v_j) + \sum_{-(ij)} w_{ij}(1-v_i\cdot v_j)\\&\text{subject to}\\&\quad v_i\cdot v_i = 1 \quad \text{for all } i\\&\quad v_i\cdot v_j \ge 0 \quad \text{for all } i,j\end{aligned} \tag{4}$$

In this case we interpret $v_i \cdot v_j = 1$ as evidence that i and j are in the same cluster, but $v_i \cdot v_j = 0$ as evidence toward separation.

Emanuel and Fiat (2003) extended the work of Bansal et al. by drawing a link between Correlation Clustering and the Minimum Multicut problem. This reduction to Multicut provided an $O(\log n)$ approximation algorithm for minimizing general instances of Correlation Clustering. Interestingly, Emanuel and Fiat also showed that there was reduction in the opposite direction: an optimal solution to Correlation Clustering induced an optimal solution to Minimum Multicut.

Demaine and Immorlica (2003) also drew the link from Correlation Clustering to Minimum multicut and its $O(\log n)$ approximation algorithm. In addition, they described an $O(r^3)$-approximation algorithm for graphs that exclude the complete bipartite graph $K_{r,r}$ as a minor.

Swamy (2004), using the same SDP (4) as Charikar et al., but different rounding techniques, showed how to maximize (2) within factor 0.7666 in general instances.

The factor 4 approximation for minimization (1) of complete instances was lowered to 2.5 by Ailon et al. (2005). Using the *distances* obtained by solving the linear program (3), they repeat the following steps:

> form a cluster around random item i by including each (unclustered) j with probability $1 - x_{ij}$; set the cluster aside.

Since solving the linear program is highly resource hungry, Ailon et al. provided a combinatorial alternative: add j to i's cluster if $w_{ij}^{+} > w_{ij}^{-}$. Not only is this algorithm very fast, it is actually a factor 3 approximation.

Recently, Tan (2007) has shown that the $79/80 + \epsilon$ inapproximability for maximizing (2) on general weighted graphs extends to general unweighted graphs.

A further variant in the Correlation Clustering family of problems is the maximization of (2)–(1), known as *maximizing correlation*. Charikar and Wirth (2004) proved an $\Omega(1/\log n)$ approximation for the general problem of maximizing

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_i x_j, \qquad \text{s.t } x_i \in \{-1,1\} \text{ for all } i, \tag{5}$$

for a matrix A with null diagonal entries, by rounding the canonical SDP relaxation. This effectively maximized correlation with the requirement that two clusters be formed; it was not hard to extend this to general instances. The gap between the vector SDP solution and the integral solution to maximizing the quadratic program (5) was in fact shown to be $\Theta(1/\log n)$ in general (Alon et al. 2006). However, in other instances such as those with a bounded number of nonzero weights for each item, a constant factor approximation was possible. Arora et al. (2005) went further and showed that it is *quasi*-NP-hard to approximate the maximization to a factor better than $\Omega(1/\log^{\gamma} n)$ for some $\gamma > 0$.

Shamir et al. (2004) showed that ▸ Cluster Editing and p-Cluster Editing, in which p clusters must be formed, are NP-complete (for $p \geq 2$). Gramm et al. (2004) took an innovative approach to solving the Clustering Editing problem exactly. They had previously produced an $O(2.27k + n^3)$ time hand-made search tree algorithm, where k is the number of edges that need to be modified. This "awkward and error-prone work" was then replaced with a computer program that itself designed a search tree algorithm, involving automated case analysis, that ran in $O(1.92^k + n^3)$ time.

Kulis et al. (2005) unify various forms of clustering, correlation clustering, spectral clustering, and clustering with constraints in their kernel-based approach to k-means. In this, they have a general objective function that includes penalties for violating pairwise constraints and for having points spread far apart from their cluster centers, where the *spread* is measured in some high-dimensional space.

Applications

The work of Demaine and Immorlica (2003) on Correlation Clustering was closely linked with that of Bejerano et al. on Location Area Planning. This problem is concerned with the allocation of cells in a cellular network to clusters known as *location areas*. There are costs associated with traffic between the location areas (cuts between clusters) and with the size of clusters themselves (related to paging phones within individual cells). These costs drive the clustering solution in opposite directions, on top of which there are constraints on cells that must (or cannot) be in the same cluster. The authors show that the same $O(\log n)$ region-growing algorithm for minimizing Correlation Clustering and Multicut applies to Location Area Planning.

Correlation clustering has been directly applied to the coreference problem in natural language processing and other instances in which there are multiple references to the same object (Daume 2006; McCallum and Wellner 2005). Assuming some sort of undirected graphical model, such as a Conditional Random Field, algorithms for correlation clustering are used to partition a graph whose edge weights corresponding to log-potentials between node pairs. The machine learning community has applied some of the algorithms for Correlation clustering to problems such as email clustering and image segmentation. With similar applications in mind, Finley and Joachims (2005) explore the idea of adapting the pairwise input information to fit example clusterings given by a user. Their objective function is the same as Correlation Clustering (2), but their main tool is the ▸ Support Vector Machine.

There has been considerable interest in the ▸ consensus clustering problem, which is an excellent application of Correlation clustering techniques. Gionis et al. (2005) note several sources of motivation for the Consensus Clustering; these include identifying the correct number of clusters and improving clustering robustness. They adapt Charikar et al.'s region-growing algorithm to create a three-approximation that performs reasonably well in practice, though not as well as local search techniques. Gionis et al. also suggest using sampling as a tool for handling large data sets. Bertolacci and Wirth (2007) extended this study by implementing Ailon et al.'s algorithms with sampling, and therefore a variety of ways of developing a full clustering from the clustering of the sample. They noted that LP-based methods performed best, but placed a significant strain on resources.

Applications of Clustering with Advice

The ► *k*-means clustering algorithm is perhaps the most-used clustering technique: Wagstaff et al. incorporated constraints into a highly cited *k*-means variant called COP-KMEANS. They applied this algorithm to the task of identifying lanes of traffic based on input GPS data.

In the constrained-clustering framework, the constraints are usually assumed to be consistent (noncontradictory) and hard. In addition to the usual must- and cannot-link constraints, Davidson and Ravi (2005) added constraints enforcing various requirements on the distances between points in particular clusters. They analyzed the computational feasibility of the problem of establishing the (in) feasibility of a set of constraints, for various constraint types. Their constrained *k*-means algorithms were used to help a robot discover objects in a scene.

Recommended Reading

Ailon N, Charikar M, Newman A (2005) Aggregating inconsistent information: ranking and clustering. In: Proceedings of the thirty-seventh ACM symposium on the theory of computing. ACM Press, New York, pp 684–693

Alon N, Makarychev K, Makarychev Y, Naor A (2006) Quadratic forms on graphs. Invent Math 163(3):499–522

Arora S, Berger E, Hazan E, Kindler G, Safra S (2005) On non-approximability for quadratic programs. In: Proceedings of forty-sixth symposium on foundations of computer science. IEEE Computer Society, Washington, DC, pp 206–215

Bansal N, Blum A, Chawla S (2002) Correlation clustering. In: Correlation clustering. IEEE Computer Society, Washington, DC pp 238–247

Ben-Dor A, Shamir R, Yakhini Z (1999) Clustering gene expression patterns. J Comput Biol 6: 281–297

Bertolacci M, Wirth A (2007) Are approximation algorithms for consensus clustering worthwhile? In: Proceedings of seventh SIAM international conference on data mining. SIAM, Philadelphia, pp 437–442

Charikar M, Guruswami V, Wirth A (2003) Clustering with qualitative information. In: Proceedings of forty fourth FOCS, Cambridge, pp 524–533

Charikar M, Wirth A (2004) Maximizing quadratic programs: extending Grothendieck's inequality. In: Proceedings of forty fifth FOCS, Rome, pp 54–60

Daume H (2006) Practical structured learning techniques for natural language processing. PhD thesis, University of Southern California

Davidson I, Ravi S (2005) Clustering with constraints: feasibility issues and the *k*-means algorithm. In: Proceedings of fifth SIAM international conference on data mining, Newport Beach

Demaine E, Emanuel D, Fiat A, Immorlica N (2006) Correlation clustering in general weighted graphs. Theor Comput Sci 361(2):172–187

Demaine E, Immorlica N (2003) Correlation clustering with partial information. In: Proceedings of sixth workshop on approximation algorithms for combinatorial optimization problems, pp 1–13

Emanuel D, Fiat A (2003) Correlation clustering – minimizing disagreements on arbitrary weighted graphs. In: Proceedings of eleventh European symposium on algorithms, Budapest, pp 208–220

Ferligoj A, Batagelj V (1982) Clustering with relational constraint. Psychometrika 47(4):413–426

Finley T, Joachims T (2005) Supervised clustering with support vector machines. In: Proceedings of twenty-second international conference on machine learning, Bonn

Gionis A, Mannila H, Tsaparas P (2005) Clustering aggregation. In: Proceedings of twenty-first international conference on data engineering, Tokyo

Gramm J, Guo J, Hüffner F, Niedermeier R (2004) Automated generation of search tree algorithms for hard graph modification problems. Algorithmica 39(4):321–347

Kulis B, Basu S, Dhillon I, Mooney R (2005) Semi-supervised graph clustering: a kernel approach. In: Proceedings of twenty-second international conference on machine learning, Bonn, pp 457–464

McCallum A, Wellner B (2005) Conditional models of identity uncertainty with application to noun coreference. In: Saul L, Weiss Y, Bottou L (eds) Advances in neural information processing systems 17. MIT Press, Cambridge, pp 905–912

Meilă M (2003) Comparing clusterings by the variation of information. In: Proceedings of sixteenth conference on learning theory, pp 173–187

Shamir R, Sharan R, Tsur D (2004) Cluster graph modification problems. Discr Appl Math 144:173–182

Swamy C (2004) Correlation clustering: maximizing agreements via semidefinite programming. In: Proceedings of fifteenth ACM-SIAM symposium on discrete algorithms, pp 519–520

Tan J (2007) A note on the inapproximability of correlation clustering. Technical report 0704.2092, eprint arXiv, 2007

Correlation-Based Learning

▶ Biological Learning: Synaptic Plasticity, Hebb Rule and Spike Timing Dependent Plasticity

Cost

In ▶ Markov decision processes, negative *rewards* are often expressed as *costs*. A reward of $-x$ is expressed as a cost of x. In ▶ supervised learning, *cost* is used as a synonym for ▶ loss.

Cross-References

▶ Loss

Cost Function

▶ Loss Function

Cost-Sensitive Classification

▶ Cost-Sensitive Learning

Cost-Sensitive Learning

Charles X. Ling and Victor S. Sheng
The University of Western Ontario, London, ON, Canada

Synonyms

Cost-sensitive classification; Learning with different classification costs

Definition

Cost-Sensitive Learning is a type of learning that takes the misclassification costs (and possibly other types of cost) into consideration. The goal of this type of learning is to minimize the total cost. The key difference between cost-sensitive learning and cost-insensitive learning is that cost-sensitive learning treats different misclassifications differently. That is, the cost for labeling a positive example as negative can be different from the cost for labeling a negative example as positive. Cost-insensitive learning does not take misclassification costs into consideration.

Motivation and Background

Classification is an important task in inductive learning and machine learning. A classifier, trained from a set of training examples with class labels, can then be used to predict the class labels of new examples. The class label is usually discrete and finite. Many effective classification algorithms have been developed, such as ▶ naïve Bayes, ▶ decision trees, ▶ neural networks, and ▶ support vector machines. However, most classification algorithms seek to minimize the error rate: the percentage of the incorrect prediction of class labels. They ignore the difference between types of misclassification errors. In particular, they implicitly assume that all misclassification errors have equal cost.

In many real-world applications, this assumption is not true. The differences between different misclassification errors can be quite large. For example, in medical diagnosis of a certain cancer (where having cancer is regarded as the positive class, and non-cancer (healthy) as negative), misdiagnosing a cancer patient as healthy (the patient is actually positive but is classified as negative; thus it is also called "false negative") is much more serious (thus expensive) than a false-positive error. The patient could lose his/her life because of a delay in correct diagnosis and treatment. Similarly, if carrying a bomb is positive, then it is much more expensive to miss a terrorist who carries a bomb onto a flight than searching an innocent person.

Cost-sensitive learning takes costs, such as the misclassification cost, into consideration. Turney (2000) provides a comprehensive survey of a large variety of different types of costs in data

Cost-Sensitive Learning, Table 1 An example of cost matrix for binary classification

	Actual negative	Actual positive
Predict negative	$C(0,0)$, or *TP*	$C(0,1)$, or *FN*
Predict positive	$C(1,0)$, or FP	$C(1,1)$, or TP

mining and machine learning, including misclassification costs, data acquisition cost (instance costs and attribute costs), ▶ active learning costs, computation cost, human–computer interaction cost, and so on. The misclassification cost is singled out as the most important cost, and it has received the most attention in recent years.

Theory

The theory of cost-sensitive learning (Elkan 2001; Zadrozny and Elkan 2001) describes how the misclassification cost plays its essential role in various cost-sensitive learning algorithms.

Without loss of generality, binary classification is assumed (i.e., positive and negative class) in this paper. In cost-sensitive learning, the costs of false positive (actual negative but predicted as positive; denoted as *FP*), false negative (*FN*), true positive (*TP*), and true negative (*TN*) can be given in a cost matrix, as shown in Table 1. In the table, the notation $C(i,j)$ is also used to represent the misclassification cost of classifying an instance from its actual class j into the predicted class i (1 is used for positive, and 0 for negative). These misclassification cost values can be given by domain experts, or learned via other approaches. In cost-sensitive learning, it is usually assumed that such a cost matrix is given and known. For multiple classes, the cost matrix can be easily extended by adding more rows and more columns.

Note that $C(i,i)$ (*TP* and *TN*) is usually regarded as the "benefit" (i.e., negated cost) when an instance is predicted correctly. In addition, cost-sensitive learning is often used to deal with datasets with very imbalanced class distributions (see ▶ Class Imbalance Problem) (Japkowicz and Stephen 2002). Usually (and without loss of generality), the minority or rare class is regarded as the positive class, and it is often more expensive to misclassify an actual positive example into negative, than an actual negative example into positive. That is, the value of $FN = C(0,1)$ is usually larger than that of $FP = C(1,0)$. This is true for the cancer example mentioned earlier (cancer patients are usually rare in the population, but predicting an actual cancer patient as negative is usually very costly) and the bomb example (terrorists are rare).

Given the cost matrix, an example should be classified into the class that has the minimum expected cost. This is the minimum expected cost principle. The expected cost $R(i|x)$ of classifying an instance x into class i (by a classifier) can be expressed as:

$$R(i|x) = \sum_j P(j|x)C(j,i), \qquad (1)$$

where $P(j|x)$ is the probability estimation of classifying an instance into class j. That is, the classifier will classify an instance x into positive class if and only if:

$$P(0|x)C(1,0) + P(1|x)C(1,1) \leq P(0|x)C(0,0) \\ + P(1|x)C(0,1)$$

This is equivalent to:

$$P(0|x)C(1,0) - C(0,0) \leq P(1|x) \\ (C(0,1) - C(1,1))$$

Thus, the decision (of classifying an example into positive) will not be changed if a constant is added into a column of the original cost matrix. Thus, the original cost matrix can always be converted to a simpler one by subtracting $C(0,0)$ to the first column, and $C(1,1)$ to the second column. After such conversion, the simpler cost matrix is shown in Table 2. Thus, any given cost-matrix can be converted to one with $C(0,0) = C(1,1) = 0$. (Here it is assumed that the misclassification cost is the same for all examples. This property is a special case of the one discussed in Elkan (2001).) In the rest of the paper, it will be assumed that $C(0,0) = C(1,1) = 0$. Under this

Cost-Sensitive Learning, Table 2 A simpler cost matrix with an equivalent optimal classification

	True negative	True positive
Predict negative	0	$C(0,1) - C(1,1)$
Predict positive	$C(1,0) - C(0,0)$	0

assumption, the classifier will classify an instance x into positive class if and only if:

$$P(0|x)C(1,0) \leq P(1|x)C(0,1)$$

As $P(0|x) = 1 - P(1|x)$, a threshold p^* can be obtained for the classifier to classify an instance x into positive if $P(1|x) \geq p^*$, where

$$P^* = \frac{C(1,0)}{C(1,0) + C(0,1)}. \qquad (2)$$

Thus, if a cost-insensitive classifier can produce a posterior probability estimation $p(1|x)$ for each test example x, one can make the classifier cost-sensitive by simply choosing the classification threshold according to (2), and classify any example to be positive whenever $P(1|x) \geq p^*$. This is what several cost-sensitive meta-learning algorithms, such as *Relabeling*, are based on (see later for details). However, some cost-insensitive classifiers, such as C4.5, may not be able to produce accurate probability estimation; they return a class label without a probability estimate. *Empirical Thresholding* (Sheng and Ling 2006) does not require accurate estimation of probabilities – an accurate ranking is sufficient. It simply uses ▸ cross-validation to search for the best probability value $p*$ to use as a threshold.

Traditional cost-insensitive classifiers are designed to predict the class in terms of a default, fixed threshold of 0.5. Elkan (2001) shows that one can "rebalance" the original training examples by sampling, such that the classifiers with the 0.5 threshold is equivalent to the classifiers with the p^* threshold as in (2), in order to achieve cost-sensitivity. The rebalance is done as follows. If all positive examples (as they are assumed as the rare class) are kept, then the number of negative examples should be multiplied by $C(1,0)/C(0,1) = FP/FN$. Note that as usualy $FP < FN$, the multiple is less than 1. This is, thus, often called "under-sampling the majority class." This is also equivalent to "proportional sampling," where positive and negative examples are sampled by the ratio of:

$$p(1)FN : p(0)FP \qquad (3)$$

where $p(1)$ and $p(0)$ are the prior probability of the positive and negative examples in the original training set. That is, the prior probabilities and the costs are interchangeable: doubling $p(1)$ has the same effect as doubling *FN*, or halving *FP* (Drummond and Holte 2000). Most sampling meta-learning methods, such as costing (Zadrozny et al. 2003), are based on (3) above (see later for details).

Almost all meta-learning approaches are either based on (2) or (3) for the thresholding- and sampling-based meta-learning methods, respectively, to be discussed in the next section.

Structure of Learning System

Broadly speaking, cost-sensitive learning can be categorized into two categories. The first one is to design classifiers that are cost-sensitive in themselves. They are called the direct method. Examples of direct cost-sensitive learning are ICET (Turney 1995) and cost-sensitive decision tree (Drummond and Holte 2000; Ling et al. 2004). The other category is to design a "wrapper" that converts any existing cost-insensitive (or cost-blind) classifiers into cost-sensitive ones. The wrapper method is also called cost-sensitive meta-learning method, and it can be further categorized into thresholding and sampling. Here is a hierarchy of the cost-sensitive learning and some typical methods. This paper will focus on cost-sensitive meta-learning that considers the misclassification cost only.

Cost-Sensitive learning

– Direct methods
  - ICET (Turney 1995)
  - Cost-sensitive decision trees (Drummond and Holte 2000; Ling et al. 2004)

- Meta-learning
  - Thresholding
    - MetaCost (Domingos 1999)
    - CostSensitiveClassifier (CSC in short) (Witten and Frank 2005)
    - Cost-sensitive naïve Bayes (Chai et al. 2004)
    - Empirical Thresholding (ET in short) (Sheng and Ling 2006)
  - Sampling
    - Costing (Zadrozny et al. 2003)
    - Weighting (Ting 1998)

Direct Cost-Sensitive Learning

The main idea of building a direct cost-sensitive learning algorithm is to directly introduce and utilize misclassification costs into the learning algorithms. There are several works on direct cost-sensitive learning algorithms, such as ICET (Turney 1995) and cost-sensitive decision trees (Ling et al. 2004).

ICET (Turney 1995) incorporates misclassification costs in the fitness function of genetic algorithms. On the other hand, cost-sensitive decision tree (Ling et al. 2004), called CSTree here, uses the misclassification costs directly in its tree building process. That is, instead of minimizing entropy in attribute selection as in C4.5, CSTree selects the best attribute by the expected total cost reduction. That is, an attribute is selected as a root of the (sub) tree if it minimizes the total misclassification cost.

Note that as both ICET and CSTree directly take costs into model building, they can also take easily attribute costs (and perhaps other costs) directly into consideration, while meta cost-sensitive learning algorithms generally cannot.

Drummond and Holte (2000) investigate the cost-sensitivity of the four commonly used attribute selection criteria of decision tree learning: accuracy, Gini, entropy, and DKM. They claim that the sensitivity of cost is highest with the accuracy, followed by Gini, entropy, and DKM.

Cost-Sensitive Meta-Learning

Cost-sensitive meta-learning converts existing cost- insensitive classifiers into cost-sensitive ones without modifying them. Thus, it can be regarded as a middleware component that preprocesses the training data, or post-processes the output, from the cost-insensitive learning algorithms.

Cost-sensitive meta-learning can be further classified into two main categories: *thresholding* and *sampling*, based on (2) and (3) respectively, as discussed in the theory section.

Thresholding uses (2) as a threshold to classify examples into positive or negative if the cost-insensitive classifiers can produce probability estimations. *MetaCost* (Domingos 1999) is a *thresholding* method. It first uses bagging on decision trees to obtain reliable probability estimations of training examples, relabels the classes of training examples according to (2), and then uses the relabeled training instances to build a cost-insensitive classifier. *CSC* (Witten and Frank 2005) also uses (2) to predict the class of test instances. More specifically, *CSC* uses a cost-insensitive algorithm to obtain the probability estimations $P(j|x)$ of each test instance. (CSC is a meta-learning method and can be applied to any classifiers.) Then it uses (2) to predict the class label of the test examples. Cost-sensitive naïve Bayes (Chai et al. 2004) uses (2) to classify test examples based on the posterior probability produced by the naïve Bayes.

As seen, all *thresholding*-based meta-learning methods rely on accurate probability estimations of $p(1|x)$ for the test example x. To achieve this, Zadrozny and Elkan (2001) propose several methods to improve the calibration of probability estimates. *ET* (Empirical Thresholding) (Sheng and Ling 2006) is a thresholding-based meta-learning method. It does not require accurate estimation of probabilities – an accurate ranking is sufficient. *ET* simply uses cross-validation to search the best probability from the training instances as the threshold, and uses the searched threshold to predict the class label of test instances.

On the other hand, *sampling* first modifies the class distribution of the training data according to (3), and then applies cost-insensitive classifiers on the sampled data directly. There is no need for the classifiers to produce probability estimations, as long as they can classify positive or negative examples accurately. Zadrozny et al. (2003) show that proportional sampling with replacement produces duplicated cases in the training, which in turn produces overfitting in model building. Instead, Zadrozny et al. (2003) proposes to use "rejection sampling" to avoid duplication. More specifically, each instance in the original training set is drawn once, and accepted into the sample with the accepting probability $C(j,i)/Z$, where $C(j,i)$ is the misclassification cost of class i, and Z is an arbitrary constant such that $Z \geq \max C(j,i)$. When $Z = \max_{ij} C(j,i)$, this is equivalent to keeping all examples of the rare class, and sampling the majority class without replacement according to $C(1,0)/C(0,1)$ – in accordance with (3). Bagging is applied after rejection sampling to improve the results further. The resulting method is called *Costing*.

Weighting (Ting 1998) can also be viewed as a sampling method. It assigns a normalized weight to each instance according to the misclassification costs specified in (3). That is, examples of the rare class (which carries a higher misclassification cost) are assigned, proportionally, high weights. Examples with high weights can be viewed as example duplication – thus oversampling. *Weighting* then induces cost-sensitivity by integrating the instances' weights directly into C4.5, as C4.5 can take example weights directly in the entropy calculation. It works whenever the original cost-insensitive classifiers can accept example weights directly. (Thus, it can be said that *Weighting* is a semi meta-learning method.) In addition, *Weighting* does not rely on bagging as *Costing* does, as it "utilizes" all examples in the training set.

Recommended Reading

Chai X, Deng L, Yang Q, Ling CX (2004) Test-cost sensitive naïve Bayesian classification. In: Proceedings of the fourth IEEE international conference on data mining. IEEE Computer Society Press, Brighton

Domingos P (1999) MetaCost: a general method for making classifiers cost-sensitive. In: Proceedings of the fifth international conference on knowledge discovery and data mining, San Diego. ACM, New York, pp 155–164

Drummond C, Holte R (2000) Exploiting the cost (in)sensitivity of decision tree splitting criteria. In: Proceedings of the 17th international conference on machine learning, Stanford, pp 239–246

Elkan C (2001) The foundations of cost-sensitive learning. In: Proceedings of the 17th international joint conference of artificial intelligence. Morgan Kaufmann, Seattle, pp 973–978

Japkowicz N, Stephen S (2002) The class imbalance problem: a systematic study. Intell Data Anal 6(5):429–450

Ling CX, Yang Q, Wang J, Zhang S (2004) Decision trees with minimal costs. In: Proceedings of 2004 international conference on machine learning (ICML'2004), Banff

Sheng VS, Ling CX (2006) Thresholding for making classifiers cost-sensitive. In: Proceedings of the 21st national conference on artificial intelligence, 16–20 July 2006, Boston, pp 476–481

Ting KM (1998) Inducing cost-sensitive trees via instance weighting. In: Proceedings of the second European symposium on principles of data mining and knowledge discovery. Springer, Heidelberg, pp 23–26

Turney PD (1995) Cost-sensitive classification: empirical evaluation of a hybrid genetic decision tree induction algorithm. J Artif Intell Res 2:369–409

Turney PD (2000) Types of cost in inductive concept learning. In: Proceedings of the workshop on cost-sensitive learning at the 17th international conference on machine learning, Stanford University, Stanford

Witten IH, Frank E (2005) Data mining – practical machine learning tools and techniques with Java implementations. Morgan Kaufmann, San Francisco

Zadrozny B, Elkan C (2001) Learning and making decisions when costs and probabilities are both unknown. In: Proceedings of the seventh international conference on knowledge discovery and data mining, pp 204–213

Zadrozny B, Langford J, Abe N (2003) Cost-sensitive learning by cost-proportionate instance weighting. In: Proceedings of the third international conference on data mining, Melbourne

Cost-to-Go Function Approximation

▶ Value Function Approximation

Co-training

▶ Semi-supervised Learning

Covariance Matrix

Xinhua Zhang
NICTA, Australian National University,
Canberra, ACT, Australia
School of Computer Science, Australian
National University, Canberra, ACT, Australia
NICTA London Circuit, Canberra, ACT,
Australia

Abstract

Covariance matrix is a generalization of covariance between two univariate random variables. It is composed of the pairwise covariance between components of a multivariate random variable. It underpins important stochastic processes such as Gaussian process, and in practice it provides key characterizations between multiple random factors.

Definition

It is convenient to define a covariance matrix by using multivariate random variables (*mrv*): $\mathbf{X} = (X_1, \ldots, X_d)^\top$. For univariate random variables X_i and X_j, their covariance is defined as

$$\mathrm{Cov}(X_i, X_j) = \mathbb{E}\left[(X_i - \mu_i)\left(X_j - \mu_j\right)\right],$$

where μ_i is the mean of X_i: $\mu_i = \mathbb{E}[X_i]$. As a special case, when $i = j$, then we get the variance of X_i, $\mathrm{Var}(X_i) = \mathrm{Cov}(X_i, X_i)$. Now in the setting of *mrv*, assuming that each component random variable X_i has finite variance under its marginal distribution, the covariance matrix $\mathrm{Cov}(\mathbf{X}, \mathbf{X})$ can be defined as a d-by-d matrix whose (i, j)th entry is the covariance:

$$\begin{aligned}(\mathrm{Cov}(\mathbf{X}, \mathbf{X}))_{ij} &= \mathrm{Cov}\left(X_i, X_j\right) \\ &= \mathbb{E}\left[(X_i - \mu_i)\left(X_j - \mu_j\right)\right].\end{aligned}$$

And its inverse is also called precision matrix.

It is easy to rewrite the element-wise definition into the matrix form:

$$\mathrm{Cov}(\mathbf{X}, \mathbf{X}) = \mathbb{E}\left[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])^\top\right], \tag{1}$$

which naturally generalizes the variance of univariate random variables: $\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2]$.

Moreover, it is also straightforward to extend the covariance of a single *mrv* $\mathbf{X}$ to two *mrv*'s $\mathbf{X}$ (d dimensional) and $\mathbf{Y}$ (s dimensional), under the name cross covariance. It quantifies how much the component random variables in $\mathbf{X}$ and $\mathbf{Y}$ change together. The cross-covariance matrix is defined as a $d \times s$ matrix $\mathrm{Cov}(\mathbf{X}, \mathbf{Y})$ whose (i, j)th entry is

$$(\mathrm{Cov}(\mathbf{X}, \mathbf{Y}))_{ij} = \mathrm{Cov}(X_i, Y_j) \tag{2}$$

$$= \mathbb{E}\left[(X_i - \mathbb{E}[X_i])\left(Y_j - \mathbb{E}[Y_j]\right)\right]. \tag{3}$$

$\mathrm{Cov}(\mathbf{X}, \mathbf{Y})$ can also be written in the matrix form as

$$\mathrm{Cov}(\mathbf{X}, \mathbf{Y}) = \mathbb{E}\left[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{Y} - \mathbb{E}[\mathbf{Y}])^\top\right],$$

where the expectation is with respect to the joint distribution of $(\mathbf{X}, \mathbf{Y})$. Obviously, $\mathrm{Cov}(\mathbf{X}, \mathbf{Y})$ becomes $\mathrm{Cov}(\mathbf{X}, \mathbf{X})$ when $\mathbf{Y} = \mathbf{X}$.

Motivation and Background

The covariance between two univariate random variables measures how much they change together, and as a special case, the covariance of a random variable with itself is exactly its variance. It is important to note that covariance is an unnormalized measure of the correlation between the random variables.

As a generalization to multivariate random variables $\mathbf{X} = (X_1, \ldots, X_d)^\top$, the covariance matrix is a d-by-d matrix whose (i, j)th component is the covariance between X_i and X_j.

In many applications, it is important to characterize the relations between a set of factors, hence the covariance matrix plays an important role in practice, especially in machine learning.

Theory

Properties

Covariance $\mathrm{Cov}(\mathbf{X},\mathbf{X})$ has the following properties:

1. Positive semi-definiteness. It follows from Eq. (1) that $\mathrm{Cov}(\mathbf{X},\mathbf{X})$ is positive semi-definite. $\mathrm{Cov}(\mathbf{X},\mathbf{X}) = \mathbf{0}$ if, and only if, $\mathbf{X}$ is constant almost surely, i.e., there exists a constant $\boldsymbol{x}$ such that $\Pr(\mathbf{X} \neq \boldsymbol{x}) = 0$. $\mathrm{Cov}(\mathbf{X},\mathbf{X})$ is not positive definite if, and only if, there exists a constant $\boldsymbol{\alpha}$ such that $\langle \boldsymbol{\alpha}, \mathbf{X} \rangle$ is constant almost surely.
2. Relating cumulant to moments: $\mathrm{Cov}(\mathbf{X},\mathbf{X}) = \mathbb{E}[\mathbf{X}\mathbf{X}^\top] - \mathbb{E}[\mathbf{X}]\mathbb{E}[\mathbf{X}]^\top$.
3. Linear transform: If $\mathbf{Y} = \mathbf{A}\mathbf{X} + \mathbf{b}$ where $\mathbf{A} \in \mathbb{R}^{s\times d}$ and $\mathbf{b} \in \mathbb{R}^s$, then $\mathrm{Cov}(\mathbf{Y},\mathbf{Y}) = \mathbf{A}\mathrm{Cov}(\mathbf{X},\mathbf{X})\mathbf{A}^\top$.

Cross-covariance $\mathrm{Cov}(\mathbf{X},\mathbf{Y})$ has the following properties:

1. Symmetry: $\mathrm{Cov}(\mathbf{X},\mathbf{Y}) = \mathrm{Cov}(\mathbf{Y},\mathbf{X})$.
2. Linearity: $\mathrm{Cov}(\mathbf{X}_1 + \mathbf{X}_2, \mathbf{Y}) = \mathrm{Cov}(\mathbf{X}_1,\mathbf{Y}) + \mathrm{Cov}(\mathbf{X}_2,\mathbf{Y})$.
3. Relating to covariance: If $\mathbf{X}$ and $\mathbf{Y}$ have the same dimension, then $\mathrm{Cov}(\mathbf{X}+\mathbf{Y}, \mathbf{X}+\mathbf{Y}) = \mathrm{Cov}(\mathbf{X},\mathbf{X}) + \mathrm{Cov}(\mathbf{Y},\mathbf{Y}) + 2\mathrm{Cov}(\mathbf{Y},\mathbf{X})$.
4. Linear transform: $\mathrm{Cov}(\mathbf{A}\mathbf{X}, \mathbf{B}\mathbf{Y}) = \mathbf{A}\mathrm{Cov}(\mathbf{X}, \mathbf{Y})\mathbf{B}$.

It is highly important to note that $\mathrm{Cov}(\mathbf{X},\mathbf{Y}) = 0$ is a necessary but not sufficient condition for $\mathbf{X}$ and $\mathbf{Y}$ to be independent.

Correlation Coefficient

Entries in the covariance matrix are sometimes presented in a normalized form by dividing each entry by its corresponding standard deviations. This quantity is called the correlation coefficient, represented as ρ_{X_i,X_j}, and defined as

$$\rho_{X_i,X_j} = \frac{\mathrm{Cov}(X_i,X_j)}{\mathrm{Cov}(X_i,X_i)^{1/2}\mathrm{Cov}(X_j,X_j)^{1/2}}.$$

The corresponding matrix is called the correlation matrix, and for Γ_X set to $\mathrm{Cov}(\mathbf{X},\mathbf{X})$ with all non-diagonal entries zeroed, and Γ_Y likewise, then the correlation matrix is given by

$$\mathrm{Corr}(\mathbf{X},\mathbf{Y}) = \Gamma_X^{-1/2}\mathrm{Cov}(\mathbf{X},\mathbf{Y})\Gamma_Y^{-1/2}.$$

The correlation coefficient takes on values between $[-1, 1]$.

Parameter Estimation

Given observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ of a *mrv* $\mathbf{X}$, an unbiased estimator of $\mathrm{Cov}(\mathbf{X},\mathbf{X})$ is

$$S = \frac{1}{n-1}\sum_{i=1}^{n}(\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^\top,$$

where $\bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i$. The denominator $n-1$ reflects the fact that the mean is unknown and the sample mean is used in place. Note the maximum likelihood estimator in this case replaces the denominator $n-1$ by n.

Conjugate Priors

Covariance matrix is used to define the Gaussian distribution. In this case, the inverse Wishart distribution is the conjugate prior for the covariance matrix. Since the gamma distribution is a 1-D version of the Wishart distribution, hence in the 1-D case, the gamma is the conjugate prior for precision matrix.

Applications

Several key uses of the covariance matrix are reviewed here.

Correlation and Least Squares Approximation

In many machine learning problems, we often need to quantify the correlation of two *mrv*s which may be from two different spaces. For example, we may want to study how much the image stream of a movie is correlated with the comments it receives. For simplicity, we consider a r-dimensional *mrv* $\mathbf{X}$ and a s-dimensional *mrv* $\mathbf{Y}$. To study their correlation, suppose we have n pairs of observations $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$ drawn *iid* from certain underlying joint distribution of $(\mathbf{X}, \mathbf{Y})$. Let $\bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^n \boldsymbol{x}_i$ and $\bar{\boldsymbol{y}} = \frac{1}{n}\sum_{i=1}^n \boldsymbol{y}_i$ and stack $\{\boldsymbol{x}_i\}$ and $\{\boldsymbol{y}_i\}$ into $\tilde{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^\top$ and $\tilde{Y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)^\top$, respectively. Then the cross-covariance matrix $\mathrm{Cov}(\mathbf{X}, \mathbf{Y})$ can be estimated by $\frac{1}{n}\sum_{i=1}^n (\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{y}_i - \bar{\boldsymbol{y}})^\top$. To quantify the cross correlation by a real number, we need to apply some norm of the cross-covariance matrix, and the simplest one is the Frobenius norm: $\|A\|_F^2 = \sum_{ij} A_{ij}^2$. Therefore we obtain a measure of cross correlation:

$$\left\| \frac{1}{n}\sum_{i=1}^n (\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{y}_i - \bar{\boldsymbol{y}})^\top \right\|_F^2 = \frac{1}{n} H\tilde{X}\tilde{X}^\top H\tilde{Y}\tilde{Y}^\top, \tag{4}$$

where $H_{ij} = \delta_{ij} - \frac{1}{n}$ and $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. It is important to notice that (a) in this measure, inner product is performed only in the space of $\mathbf{X}$ and $\mathbf{Y}$ separately, i.e., no transformation between $\mathbf{X}$ and $\mathbf{Y}$ is required, and (b) the data points affect the measure only via inner products $\boldsymbol{x}_i^\top \boldsymbol{x}_j$ as the (i, j)th entry of $\tilde{X}\tilde{X}^\top$ (and similarly for $\boldsymbol{y}_i$). Hence we can endow new inner products on $\mathbf{X}$ and $\mathbf{Y}$, which eventually allows us to apply kernels, e.g., Gretton et al. (2005). In a nutshell, kernels (Schölkopf and Smola 2002) redefine the inner product $\boldsymbol{x}_i^\top \boldsymbol{x}_j$ by mapping $\boldsymbol{x}_i$ to a richer feature space via $\phi(\boldsymbol{x}_i)$ and then compute the inner product there: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) := \phi(\boldsymbol{x}_i)^\top \phi(\boldsymbol{x}_j)$. Since the measure in Eq. (4) only needs inner products, one can even directly define k without explicitly specifying ϕ. This allows us to (a) implicitly use a rich feature space whose dimension can be infinitely high and (b) apply this measure of independence to non-Euclidean spaces as long as a kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ can be defined on it.

Interestingly, this measure can be equivalently motivated by least square linear regression. That is, we look for a linear transform $T : \mathbb{R}^d \to \mathbb{R}^s$ which minimizes

$$\frac{1}{n}\sum_{i=1}^n \|(\boldsymbol{y}_i - \bar{\boldsymbol{y}}) - T(\boldsymbol{x}_i - \bar{\boldsymbol{x}})\|^2 .$$

And one can show that its minimum objective value is exactly equal to Eq. (4) up to a constant, as long as all $\boldsymbol{y}_i - \bar{\boldsymbol{y}}$ and $\boldsymbol{x}_i - \bar{\boldsymbol{x}}$ have unit length. In practice, this can be achieved by normalization. Or, the measure in Eq. (4) itself can be normalized by replacing the covariance matrix with the correlation matrix.

Principal Component Analysis

The covariance matrix plays a key role in principal component analysis (PCA). Assume we are given n *iid* observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ of a *mrv* $\mathbf{X}$, and let $\bar{\boldsymbol{x}} = \frac{1}{n}\sum_i \boldsymbol{x}_i$. PCA tries to find a set of orthogonal directions $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots$, such that the projection of $\mathbf{X}$ to the direction $\boldsymbol{w}_1$, $\boldsymbol{w}_1^\top \mathbf{X}$ has the highest variance among all possible directions in the d-dimensional space. After subtracting from $\mathbf{X}$ the projection to $\boldsymbol{w}_1$, $\boldsymbol{w}_2$ is chosen as the highest variance projection direction for the remainder. This procedure goes on, giving $\boldsymbol{w}_3, \ldots, \boldsymbol{w}_d$.

To find $\boldsymbol{w}_1 := \mathrm{argmax}_{\boldsymbol{w}} \mathrm{Var}(\boldsymbol{w}^\top \mathbf{X})$, we need an empirical estimate of $\mathrm{Var}(\boldsymbol{w}^\top \mathbf{X})$. Estimating $\mathbb{E}[(\boldsymbol{w}^\top \mathbf{X})^2]$ by $\boldsymbol{w}^\top \left(\frac{1}{n}\sum_i \boldsymbol{x}_i \boldsymbol{x}_i^\top\right) \boldsymbol{w}$ and $\mathbb{E}[\boldsymbol{w}^\top \mathbf{X}]$ by $\frac{1}{n}\sum_i \boldsymbol{w}^\top \boldsymbol{x}_i$, we get

$$\boldsymbol{w}_1 = \mathrm{argmax}_{\boldsymbol{w}:\|\boldsymbol{w}_1\|=1} \boldsymbol{w}^\top S \boldsymbol{w}, \quad \text{where}$$

$$S = \frac{1}{n}\sum_{i=1}^n (\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^\top,$$

i.e., S is $\frac{n}{n-1}$ times the unbias empirical estimate of the covariance of $\mathbf{X}$, based on samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. $\boldsymbol{w}_1$ turns out to be exactly the eigenvector of S corresponding to the greatest eigenvalue.

Note that PCA is independent of the distribution of $\mathbf{X}$. More details on PCA can be found at Jolliffe (2002).

Gaussian Processes

Gaussian processes are another important framework in machine learning that relies on the covariance matrix. It is a distribution over functions $f(\cdot)$ from certain space $\mathcal{X}$ to $\mathbb{R}$, such that for any $n \in \mathbb{N}$ and any n points $\{\boldsymbol{x}_i \in \mathcal{X}\}_{i=1}^n$, the set of values of f evaluated at $\{\boldsymbol{x}_i\}_i$, $\{f(x_1), \ldots, f(x_n)\}$, will have an n-dimensional Gaussian distribution. Different choices of the covariance matrix of the multivariate Gaussian lead to different stochastic processes such as

Wiener process, Brownian motion, Ornstein-Uhlenbeck process, etc. In these cases, it makes more sense to define a covariance function $C : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$, such that given any set $\{\boldsymbol{x}_i \in \mathcal{X}\}_{i=1}^n$ for any $n \in \mathbb{N}$, the n-by-n matrix $\left(C(\boldsymbol{x}_i, \boldsymbol{x}_j)\right)_{ij}$ is positive semi-definite and can be used as the covariance matrix. This further allows straightforward kernelization of a Gaussian process by using the kernel function as the covariance function.

Although the space of functions is infinite dimensional, the marginalization property of multivariate Gaussian distributions guarantees that the user of the model only needs to consider the observed $\boldsymbol{x}_i$ and ignore all the other possible $\boldsymbol{x} \in \mathcal{X}$. This important property says that for a *mrv* $\mathbf{X} = (\mathbf{X}_1^\top, \mathbf{X}_2^\top)^\top \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, the marginal distribution of $\mathbf{X}_1$ is $\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_{11})$, where Σ_{11} is the submatrix of Σ corresponding to $\mathbf{X}_1$ (and similarly for $\boldsymbol{\mu}_1$). So taking into account the random variable $\mathbf{X}_2$ will not change the marginal distribution of $\mathbf{X}_1$.

Cross-References

▸ Gaussian Distribution
▸ Gaussian Processes

Recommended Reading

For a complete treatment of covariance matrix from a statistical perspective, see Casella and Berger (2002) and Mardia et al. (1979) provides details for the multivariate case. PCA is comprehensively discussed in Jolliffe (2002), and kernel methods are introduced in Schölkopf and Smola (2002). Williams and Rasmussen (2006) gives the state of the art on how Gaussian processes can be utilized for machine learning.

Casella G, Berger R 2002 Statistical inference, 2nd edn. Duxbury, Pacific Grove
Gaussian Processes for Machine Learning, Carl Edward Rasmussen and Chris Williams, the MIT Press, Cambridge, MA, 2006
Gretton A, Herbrich R, Smola A, Bousquet O, Schölkopf B (2005) Kernel methods for measuring independence. J Mach Learn Res 6:2075–2129
Jolliffe IT (2002) Principal component analysis. Springer series in statistics, 2nd edn. Springer, New York
Mardia KV, Kent JT, Bibby JM (1979) Multivariate analysis. Academic, London/New York
Schölkopf B, Smola A (2002) Learning with Kernels. MIT, Cambridge

Covering Algorithm

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt,
Darmstadt, Deutschland
Department of Information Technology,
University of Leoben, Leoben, Austria

Abstract

The covering algorithm is the dominant approach to classification ▸ rule learning. Its distinguishing feature is the idea to learn one rule at a time, successively removing all training examples that are covered by the learned rules.

Synonyms

Separate-and-conquer learning

Method

Most covering algorithms operate in a ▸ concept learning framework, i.e., they assume a set of positive and negative training examples. Adaptations to the multi-class case are typically performed via ▸ class binarization, learning different ▸ rule sets for binary problems. Some algorithms, most notably CN2 (Clark and Niblett 1989; Clark and Boswell 1991), learn multi-class rules directly by optimizing overall possible classes in the head of the rule. In this case, the resulting theory is interpreted as a ▸ decision list. In the following, we will assume a two-class problem with a positive and a negative class.

The Covering algorithm starts with an empty theory. If there are any positive examples in the training set, it calls the subroutine FindBestRule for learning a single rule that

```
procedure COVERING(Examples, Classifier)

Input: Examples, a set of positive and negative examples
for a class $c$.

// initialize the classifier
$Classifier = \emptyset$

//loop until no more positive examples are covered
while POSITIVE(Examples) $\neq \emptyset$ do

    // find the best rule for the current examples
    $Rule$ = FINDBESTRULE(Examples)

    // check if we need more rules
    if RULESTOPPINGCRITERION(Classifier,Rule,
      Examples)
    then  break while

    // remove covered examples and add rule to rule set
    $Examples = Examples \setminus$ COVER(Rule,Examples)
    $Classifier = Classifier \cup Rule$

endwhile

// post-process the rule set (e.g., pruning)
$Classifier$ = POSTPROCESSING (Classifier)

Output: Classifier
```

will cover a subset of the positive examples (and possibly some negative examples as well). All covered examples are then separated from the training set, the learned rule is added to the theory, and another rule is learned from the remaining examples. Rules are learned in this way until no positive examples are left or until the RULESTOPPINGCRITERION fires. In the simplest case, the stopping criterion is a check whether there are still remaining positive examples that need to be covered. The resulting theory may also undergo some POSTPROCESSING, e.g., a separate pruning and re-induction phase as in RIPPER (Cohen 1995).

A more extensive survey of this family of algorithms can be found in Fürnkranz (1999a).

Cross-References

- ► Class Binarization
- ► Concept Learning
- ► Decision List
- ► Rule Learning
- ► Rule Set

Recommended Reading

Clark P, Boswell R (1991) Rule induction with CN2: some recent improvements. In: Proceedings of the 5th European working session on learning (EWSL-91), Porto. Springer, pp 151–163

Clark P, Niblett T (1989) The CN2 induction algorithm. Mach Learn 3(4):261–283

Cohen WW (1995) Fast effective rule induction. In: Prieditis A, Russell S (eds) Proceedings of the 12th international conference on machine learning (ML-95), Lake Tahoe. Morgan Kaufmann, pp 115–123

Fürnkranz J (1999) Separate-and-conquer rule learning. Artif Intell Rev 13(1):3–54. http://www.ofai.at/cgi-bin/tr-online?number+96-25

Credit Assignment

Claude Sammut
The University of New South Wales, Sydney,
NSW, Australia

Synonyms

Structural credit assignment; Temporal credit assignment

Definition

When a learning system employs a complex decision process, it must assign credit or blame for the outcomes to each of its decisions. Where it is not possible to directly attribute an individual outcome to each decision, it is necessary to apportion credit and blame between each of the combinations of decisions that contributed to the outcome. We distinguish two cases in the credit assignment problem. *Temporal credit assignment* refers to the assignment of credit for outcomes to actions. *Structural credit assignment* refers to the assignment of credit for actions to internal decisions. The first subproblem involves determining when the actions that deserve credit were taken and the second involves assigning credit to the internal structure of actions (Sutton 1984).

Motivation

Consider the problem of learning to balance a pole that is hinged on a cart (Michie and Chambers 1968; Anderson and Miller 1991). The cart is constrained to run along a track of finite length and a fixed force can be applied to push the cart left or right. A controller for the pole and cart system must make a decision whether to push left or right at frequent, regular time intervals, for example, 20 times a second. Suppose that this controller is capable of learning from trial-and-error. If the pole falls over, then it must determine which actions it took helped or hurt its performance. Determining that action is the problem of *temporal credit assignment*. Although the actions are directly responsible for the outcome of a trial, the internal process for choosing the action indirectly affects the outcome. Assigning credit or blame to those internal processes that lead to the choice of action is the *structural credit assignment* problem. In the case of pole balancing, the learning system will typically keep statistics such as how long, on average, the pole remained balanced after taking a particular action in a particular state, or after a failure, it may count back and determine the average amount of time to failure after taking a particular action in a particular state. Using these statistics, the learner attempts to determine the best action for a given state.

The above example is typical of many problems in ▸ reinforcement learning (Sutton and Barto 1998), where an agent interacts with its environment and through that interaction, learns to improve its performance in a task. Although Samuel (1959) was the first to use a form of reinforcement learning in his checkers playing program, Minsky (1961) first articulated the credit assignment, as follows:

> Using devices that also learn which events are associated with reinforcement, i.e., reward, we can build more autonomous "secondary reinforcement" systems. In applying such methods to complex problems, one encounters a serious difficulty – in distributing credit for success of a complex strategy among the many decisions that were involved.

The BOXES algorithm of Michie and Chambers (1968) learned to control a pole balancer and performed credit assignment but the problem of credit assignment later became central to reinforcement learning, particularly following the work of Sutton (1984). Although credit assignment has become most strongly identified with reinforcement learning, it may appear in any learning system that attempts to assess and revise its decision-making process.

Structural Credit Assignment

The setting for our learning system is that we have an agent that interacts with an environment. The environment may be a virtual one, as in game playing, or it may be physical, as in a robot performing some task. The agent receives input, possibly through sensing devices, that allows it to characterize the state of the world. Somehow, the agent must map these inputs to appropriate responses. These responses may change the state of the world. In reinforcement learning, we assume that the agent will receive some reward signal after an action or sequence of actions. Its job is to maximize these rewards over time.

Structural credit assignment is associated with generalization over the input space of the agent. For example, a game player may have to respond to a very large number of potential board positions or a robot may have to respond to a stream of camera images. It is infeasible to learn a complete mapping from every possible input to every possible output. Therefore, a learning agent will typically use some means of grouping input signals. In the case of the BOXES pole balancer, Michie and Chambers discretized the state space. The state is characterized by the cart's position and velocity and the pole's angle and angular velocity. These parameters create a four-dimensional space, which was broken into three regions (left, center, right) for the pole angle, five for the angular velocity, and three for the cart position and velocity. These choices were arbitrary and other combinations also worked.

Having divided the input space into non-overlapping regions, Michie and Chambers associated a push-left and push-right action with each region, or box. The learning algorithm maintains a score for each action and chooses the next action based on that score. BOXES was an early, and simple example, of creating an internal representation for mapping inputs to outputs. The problem with this method is that the structure of the decision-making system is fixed at the start and the learner is incapable of changing the representation. This may be needed if, for example, the subdivisions that were chosen do not correspond to a real decision boundary. A learning system that could adapt its representation has an advantage, in this case.

The BOXES representation can be thought of as a lookup table that implements a function that maps an input to an output. The fixed lookup table can be replaced by a function approximator that, given examples from the desired function, generalizes from them to construct an approximation of that function. Different function approximation techniques can be used. For example, Moore's (1990) function approximator was a ▶ nearest-neighbor algorithm, implemented using kd-tree to improve efficiency. Other function approximation methods may also be used, e.g., Albus' CMAC algorithm (1975), ▶ locally weighted regression (Atkeson et al. 1997), ▶ perceptrons Rosenblatt (1962), multi-layer networks (Hinton et al. 1985), ▶ radial basis functions, etc. Structural credit assignment is also addressed in the creation of hierarchical representations. See ▶ hierarchical reinforcement learning. Other approaches to structural credit assignment include ▶ Value function approximation (Bertsekas and Tsitsiklis 1996) and automatic basis generation (Mahadevan 2009). See the entry on ▶ Gaussian Processes for examples of recent Bayesian and kernel method based approaches to solving the credit assignment problem.

Temporal Credit Assignment

In the pole balancing example described above, the learning system receives a signal when the pole has fallen over. How does it know which actions leading up to the failure contributed to the fall? The system will receive a high-level punishment in the event of a failure or a reward in tasks where there is a goal to be achieved. In either case, it makes sense to assign the greatest credit or blame to the most recent actions and assign progressively less to the preceding actions. Each time a learning trial is repeated, the value of an action is updated so that if it leads to another action of higher value, its weight is increased. Thus, the reward or punishment propagates back through the sequence of decisions taken by the system. The credit assignment problem was addressed by Michie and Chambers, in the BOXES, algorithm but many other solutions have subsequently been proposed. See the entries on ▶ Q-learning (Watkins 1989, 1992) and ▶ temporal difference learning (Barto et al. 1983; Sutton 1984).

Although temporal credit assignment is usually associated with reinforcement learning, it also appears in other forms of learning. In learning by imitation or ▶ behavioral cloning, an agent observes the actions of another agent and tries to learn from traces of behaviors. In this case, the learner must judge which actions of the other agent should receive credit or blame. Plan learning also encounters the same problem (Benson 1995; Wang 1996), as does ▶ explanation-based learning (Mitchell et al. 1986; Dejong and Mooney 1986; Laird et al. 1987).

To illustrate the connection with explanation-based learning, we use one of the earliest examples of this kind of learning, Mitchell and Utgoff's, LEX program (Mitchell et al. 1983). The program was intended to learn heuristics for performing symbolic integration. Given a mathematical expression that included an integral sign, the program tried to transform the expression into one they did not. The standard symbolic integration operators were known to the program but not when it is best to apply them. The task of the learning system was to learn the heuristics for when to apply the operators. This was done by experimentation. If no heuristics were available, the program attempted a brute force search. If the search was successful, all the operators applied, leading to the success were assumed to be positive examples for a heuristic, whereas operators applied during a failed attempt became negative examples. Thus, LEX performed a simple form of credit assignment, which is typical of any system that learns how to improve sequences of decisions.

▶ Genetic algorithms can also be used to evolve rules that perform sequences of actions (Holland 1986). When situation-action rules are applied in a sequence, we have a credit assignment problem that is similar to when we use a reinforcement learning. That is, how do we know which rules were responsible for success or failure and to what extent? Grefenstette (1988) describes a *bucket brigade* algorithm in which rules are given strengths that are adjusted to reflect credit or blame. This is similar to temporal difference learning except that in the bucket brigade the strengths apply to rules rather than states. See Classifier Systems and for a more comprehensive survey of bucket brigade methods, see Goldberg (1989).

Transfer Learning

After a person has learned to perform some task, learning a new, but related, task is usually easier because knowledge of the first learning episode is *transferred* to the new task. *Transfer Learning* is particularly useful for acquiring new concepts or behaviors when given only a small amount for training data. It can be viewed as a form of credit assignment because successes or failures in previous learning episodes bias future learning. Reid (2004, 2007) identifies three forms of ▶ inductive bias involved in transfer learning for rules: language bias, which determines what kinds of rules can be constructed by the learner; the search bias, which determines the order in which rules will be searched; and the evaluation bias, which determines how the quality of the rules will be assessed. Note that learning language bias is a form of structural credit assignment. Similarly, where rules are applied sequentially, evaluation bias becomes temporal credit assignment. Taylor and Stone (2009) give a comprehensive survey of transfer in ▶ reinforcement learning, in which they describe a variety of techniques for transferring the structure of an RL task from one case to another. They also survey methods for transferring evaluation bias.

Transfer learning can be applied in many different settings. Caruana (1997) developed a system for transferring inductive bias in ▶ neural networks performing multitask learning and more recent research has been directed toward transfer learning in ▶ Bayesian Networks (Niculescu and Caruana 2007).

See Transfer Learning and Silver et al. (2005) and Banerjee et al. (2006) for recent work on transfer learning.

Cross-References

▶ Bayesian Network
▶ Classifier Systems
▶ Genetic Programming
▶ Hierarchical Reinforcement Learning
▶ Inductive Bias
▶ Locally Weighted Regression for Control
▶ Nearest Neighbor
▶ Precision
▶ Radial Basis Function Networks
▶ Reinforcement Learning
▶ Temporal Difference Learning

Recommended Reading

Albus JS (1975) A new approach to manipulator control: the cerebellar model articulation controller (CMAC). J Dyn Syst Measur Control Trans ASME 97(3):220–227

Anderson CW, Miller WT (1991) A set of challenging control problems. In: Miller W, Sutton RS, Werbos PJ (eds) Neural networks for control. MIT Press, Cambridge

Atkeson C, Schaal S, Moore A (1997) Locally weighted learning. AI Rev 11:11–73

Banerjee B, Liu Y, Youngblood GM (eds) (2006) Proceedings of the ICML workshop on "structural knowledge transfer for machine learning, Pittsburgh

Barto A, Sutton R, Anderson C (1983) Neuron-like adaptive elements that can solve difficult learning control problems. IEEE Trans Syst Man Cybern SMC-13:834–846

Benson S, Nilsson NJ (1995) Reacting, planning and learning in an autonomous agent. In: Furukawa K, Michie D, Muggleton S (eds) Machine intelligence, vol 14. Oxford University Press, Oxford

Bertsekas DP, Tsitsiklis J (1996) Neuro-dynamic programming. Athena Scientific, Nashua

Caruana R (1997) Multitask learning. Mach Learn 28:41–75

Dejong G, Mooney R (1986) Explanation-based learning: an alternative view. Mach Learn 1:145–176

Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley Longman Publishing, Boston

Grefenstette JJ (1988) Credit assignment in rule discovery systems based on genetic algorithms. Mach Learn 3(2–3):225–245

Hinton G, Rumelhart D, Williams R (1985) Learning internal representation by back-propagating errors. In: Rumelhart D, McClelland J, Group TPR (eds) Parallel distributed computing: explorations in the microstructure of cognition, vol 1. MIT Press, Cambridge, pp 31–362

Holland J (1986) Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning: an artificial intelligence approach, vol 2. Morgan Kaufmann, Los Altos

Laird JE, Newell A, Rosenbloom PS (1987) SOAR: an architecture for general intelligence. Artif Intell 33(1):1–64

Mahadevan S (2009) Learning representation and control in Markov decision processes: new frontiers. Found Trends Mach Learn 1(4):403–565

Michie D, Chambers R (1968) Boxes: an experiment in adaptive control. In: Dale E, Michie D (eds) Machine intelligence, vol 2. Oliver and Boyd, Edinburgh

Minsky M (1961) Steps towards artificial intelligence. Proc IRE 49:8–30

Mitchell TM, Keller RM, Kedar-Cabelli ST (1986) Explanation based generalisation: a unifying view. Mach Learn 1:47–80

Mitchell TM, Utgoff PE, Banerji RB (1983) Learning by experimentation: acquiring and refining problem-solving heuristics. In: Michalski R, Carbonell J, Mitchell T (eds) Machine kearning: an artificial intelligence approach. Tioga, Palo Alto

Moore AW (1990) Efficient memory-based learning for robot control. Ph.D. thesis, UCAM-CL-TR-209, Computer Laboratory, University of Cambridge, Cambridge

Niculescu-mizil A, Caruana R (2007) Inductive transfer for Bayesian network structure learning. In: Proceedings of the 11th international conference on AI and statistics (AISTATS 2007), San Juan

Reid MD (2004) Improving rule evaluation using multitask learning. In: Proceedings of the 14th international conference on inductive logic programming, Porto, pp 252–269

Reid MD (2007) DEFT guessing: using inductive transfer to improve rule evaluation from limited data. Ph.D. thesis, School of Computer Science and Engineering, The University of New South Wales, Sydney

Rosenblatt F (1962) Principles of neurodynamics: perceptrons and the theory of Brain mechanics. Spartan Books, Washington, DC

Samuel A (1959) Some studies in machine learning using the game of checkers. IBM J Res Develop 3(3):210–229

Silver D, Bakir G, Bennett K, Caruana R, Pontil M, Russell S et al (2005) NIPS workshop on "inductive transfer: 10 years later", Whistler

Sutton R (1984) Temporal credit assignment in reinforcement learning. Ph.D. thesis, Department of Computer and Information Science, University of Massachusetts, Amherst

Sutton R, Barto A (1998) Reinforcement learning: an introduction. MIT Press, Cambridge

Taylor ME, Stone P (2009) Transfer learning for reinforcement learning domains: a survey. J Mach Learn Res 10:1633–1685

Wang X, Simon HA, Lehman JF, Fisher DH (1996) Learning planning operators by observation and practice. In: Proceedings of the second international conference on AI planning systems (AIPS-94), Chicago, pp 335–340

Watkins C (1989) Learning with delayed rewards. Ph.D. thesis, Psychology Department, University of Cambridge, Cambridge

Watkins C, Dayan P (1992) Q-learning. Mach Learn 8(3–4):279–292

Cross-Language Document Categorization

Document Categorization is the task consisting in assigning a document to zero, one or more categories in a predefined taxonomy. *Cross-language document categorization* describes the specific case in which one is interested in automatically categorize a document in a same taxonomy regardless of the fact that the document is written in one of several languages. For more details on the methods used to perform this task see ▶ cross-lingual text mining.

Cross-Language Information Retrieval

Cross-language information retrieval (CLIR) is the task consisting in recovering the subset of a document collection D relevant to a query q, in the special case in which D contains documents written in more than one language. Generally, it is additionally assumed that the subset of relevant documents must be returned as an ordered list, in decreasing order of relevance. For more details on methods and applications see ▶ cross-lingual text mining.

Cross-Language Question Answering

Question answering is the task consisting in finding in a document collection the answer to a question. CLCat is the specific case in which the question and the documents can be in different languages. For more details on the methods used to perform this task see ▶ cross-lingual text mining.

Cross-Lingual Text Mining

Nicola Cancedda and Jean-Michel Renders
Xerox Research Centre Europe, Meylan, France

Definition

Cross-lingual text mining is a general category denoting tasks and methods for accessing the information in sets of documents written in several languages, or whenever the language used to express an information need is different from the language of the documents. A distinguishing feature of cross-lingual text mining is the necessity to overcome some language translation barrier.

Motivation and Background

Advances in mass storage and network connectivity make enormous amounts of information easily accessible to an increasingly large fraction of the world population. Such information is mostly encoded in the form of running text which, in most cases, is written in a language different from the native language of the user. This state of affairs creates many situations in which the main barrier to the fulfillment of an information need is not technological but linguistic. For example, in some cases the user has some knowledge of the language in which the text containing a relevant piece of information is written, but does not have a sufficient control of this language to express his/her information needs. In other cases, documents in many different languages must be categorized in a same categorization schema, but manually categorized examples are available for only one language.

While the automatic translation of text from a natural language into another (machine translation) is one of the oldest problems on which computers have been used, a palette of other tasks has become relevant only more recently, due to the technological advances mentioned above. Most of them were originally motivated by needs of government Intelligence communities, but received a strong impulse from the diffusion of the World-Wide Web and of the Internet in general.

Tasks and Methods

A number of specific tasks fall under the term of Cross-lingual text mining (CLTM), including:

- *Cross-language information retrieval*
- *Cross-language document categorization*
- *Cross-language document clustering*
- *Cross-language question answering*

These tasks can in principle be performed using methods which do not involve any ▶ Text Mining, but as a matter of fact all of them have been successfully approached relying on the statistical analysis of multilingual document collections, especially *parallel corpora*. While CLTM tasks differ in many respect, they are all characterized by the fact that they require to reliably measure the similarity of two text spans written in different languages. There are essentially two families of approaches for doing this:

1. In *translation-based* approaches one of the two text spans is first translated into the language of the other. Similarity is then computed based on any measure used in mono-lingual cases. As a variant, both text spans can be translated in a third *pivot* language.
2. In *latent semantics* approaches, an abstract vector space is defined based on the statistical properties of a *parallel corpus* (or, more rarely, of a *comparable corpus*). Both text spans are then represented as vectors in such *latent semantic* space, where any similarity measure for vector spaces can be used.

The rest of this entry is organized as follows: first Translation-related approaches will be introduced, followed by Latent-semantic approaches. Finally, each of the specific CLTM tasks will be discussed in turn.

Translation-Based Approaches

The simplest approach consists in using a manually-written machine-readable bilingual dictionary: words from the first span are looked up and replaced with words in the second language (see e.g., Zhang and Vines 2005). Since typically dictionaries contain entries for "citation forms" only (e.g., the singular for nouns, the infinitive for verbs etc.), words in both spans are preliminarily *lemmatized*, i.e., replaced with the corresponding citation form. In all cases when the lexica and morphological analyzers required to perform lemmatization are not available, a frequently adopted crude alternative consists in *stemming* (i.e., truncating by taking away a suffix) both the words in the span to be translated and in the corresponding side in the lexicon. Some languages (e.g., Germanic languages) are characterized by a very productive *compounding*: simpler words are connected together to form complex words. Compound words are rarely in dictionaries as such: in order to find them it is first necessary to break compounds into their elements. This can be done based on additional linguistic resources or by means of heuristics, but in all cases it is a challenging operation in itself. If the method used afterward to compare the two spans in the target language can take weights into account, translations are "normalized" in such a way that the cumulative weight of all translations of a word is the same regardless of the number of alternative translations. Most often, the weight is simply distributed uniformly among all alternative translations. Sometimes, only the first translation for each word is kept, or the first two or three.

A second approach consists in extracting a bilingual lexicon from a *parallel corpus* instead of using a manually-written one. Methods for extracting probabilistic lexica look at the frequencies with which a word s in one language was translated with a word t to estimate the translation probability $p(t|s)$. In order to determine which word is the translation of which other word in the available examples, these examples are preliminarily aligned, first at the sentence level (to know what sentence is the translation of what other sentence) and then at the word level. Several methods for aligning sentences at the word level have been proposed, and this problem is a lively research topic in itself (see Brown et al. 1993 for a seminal paper).

Once a probabilistic bilingual dictionary is available, it can be used much in the same way as human-written dictionaries, with the notable difference that the estimated conditional probabilities provide a natural way to distribute weight across translations. When the example documents used for extracting the bilingual dictionaries are of the same style and domain as the text spans to be translated, this can result in a significant increase in accuracy for the final task, whatever this is.

It is often the case that a parallel corpus sufficiently similar in topic and style to the spans to be translated is unavailable, or it is too small to be used for reliably estimating translation probabilities. In such cases, it can be possible to replace or complement the parallel corpus with a "comparable" corpus. A comparable corpus is a pair of collections of documents, one in each of the languages of interest, which are known to be similar in content, although not the translation of one another. A typical case might be two sets of articles from corresponding sections of different newspapers collected during a same period of time. If some additional bilingual *seed* dictionary (human-written or extracted from a parallel corpus) is also available, then the comparable corpus can be leveraged as well: a word t is likely to be the translation of a word s if it turns out that the words often appearing near s are translations of the words often appearing near t. Using this observation it is thus possible to estimate the probability that t is a valid translation of s even though they are not contained in the original dictionary. Most approaches proceed by associating with s a *context vector*. This vector, with one component for each word in the source language, can simply be formed by summing together the count histograms of the words occurring within a fixed window centered in all occurrences of s in the corpus, but is often constructed using statistically more robust association measures, such as mutual information. After a possible normalization step, the context vector $CV(s)$ is translated using the seed dictionary into the target language. A context vector is also extracted from the corpus for all target words t. Eventually, a translation score between s and t is computed as $\langle Tr(CV(s)), CV(t)\rangle$:

$$\begin{aligned}\mathcal{S}(s,t) &= \langle CV(s), Tr(CV(t))\rangle \\ &= \sum_{(s',t')\in\mathcal{D}} a(s,s')a(t,t'),\end{aligned}$$

where a is the association score used to construct the context vector. While effective in many cases, this approach can provide inaccurate similarity values when polysemous words and synonyms appear in the corpus. To deal with this problem, Gaussier et al. (2004) propose the following extension:

$$\mathcal{S}(s,t) = \sum_{(s',t')\in\mathcal{D}} \left(\sum_{s'} a(s's'')a(s,s'')\right) \left(\sum_{t''} a(t',t'')a(t,t'')\right),$$

which is more robust in cases when the entries in the seed bilingual dictionary do not cover all senses actually present in the two sides of the comparable corpus.

Although these methods for building bilingual dictionaries can be (and often are) used in isolation, it can be more effective to combine them.

Using a bilingual dictionary directly is not the only way for translating a span from one language into another. A second alternative consists in using a *machine translation* (MT) system. While the MT system, in turn, relies on a bilingual dictionary of some sort, it is in general in the position of leveraging contextual clues to select the correct words and put them in the right order in the translation. This can be more or less useful depending on the specific task. MT systems fall, broadly speaking, into two classes: rule-based and statistical. Systems in the first class rely on sets of hand-written rules describing how words and syntactic structures should be translated. Statistical machine translation (SMT) systems learn this mapping by performing a statistical analysis of a parallel corpus. Some authors (e.g., Savoy and Berger 2005) also experimented with combining translation from multiple machine translation systems.

Latent Semantic Approaches

In CLTM, *Latent Semantic* approaches rely on some interlingua (language-independent) representation. Most of the time, this interlingua representation is obtained by linear or non-linear statistical analysis techniques and more specifically ▶ dimensionality reduction methods with ad-hoc optimization criterion and constraints. But, others adopt a more manual approach by exploiting multilingual thesauri or even multilingual ontologies in order to map textual objects towards a list – possibly weighted – of interlingua concepts.

For any textual object (typically a document or a section of document), the *interlingua* concept representation is derived from a sequence of operations that encompass:

1. Linguistic preprocessing (as explained in previous sections, this step amounts to extract the relevant, normalized "terms" of the textual objects, by tokenisation, word segmentation/decompounding, lemmatisation/stemming, part-of-speech tagging, stopword removal, corpus-based term filtering, Noun-phrase extractions, etc.).
2. Semantic enrichment and/or monolingual dimensionality reduction.
3. Interlingua semantic projection.

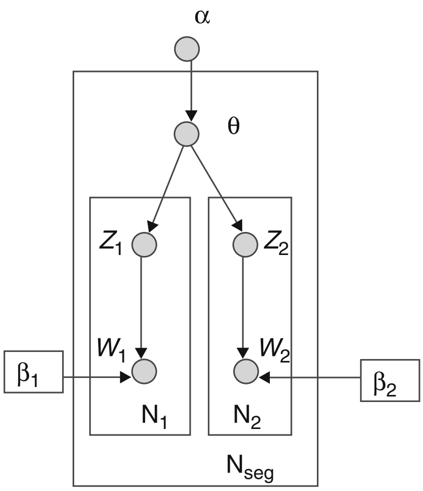
A typical semantic enrichment method is the *generalized vector space model*, that adds related terms – or neighbour terms – to each term of the textual object, neighbour terms being defined by some co-occurrence measures (for instance, mutual information). Semantic enrichment can alternatively be achieved by using (monolingual) thesaurus, exploiting relationships such as synonymy, hyperonymy and hyponymy. Monolingual dimensionality reduction consists typically in performing some *latent semantic analysis* (LSA), some form of principal component analysis on the textual object/term matrix. Dimensionality reduction techniques such as LSA or their discrete/probabilistic variants such as *probabilistic semantic analysis* (PLSA) and *latent dirichlet allocation* (LDA) offer to some extent a semantic robustness to deal with the effects of polysemy/synonymy, adopting a language-dependent concept representation in a space of dimension much smaller than the size of the vocabulary in a language.

Of course, steps (1) and (2) are highly language-dependent. Textual objects written in different languages will not follow the same linguistic processing or semantic enrichment/ dimensionality reduction. The last step (3), however, aims at projecting textual objects in the same language-independent concept space, for any source language. This is done by first extracting these common concepts, typically from a parallel corpus that offers a natural multiple-view representation of the same objects. Starting from these multiple-view observations, common factors are extracted through the use of canonical correlation analysis (CCA), cross-language latent semantic analysis, their kernelized variants (eg. Kernel-CCA) or their discrete, probabilistic extensions (cross-language latent dirichlet allocation, multinomial CCA, ...). All these methods try to discover latent factors that simultaneously explain as much as possible the "intra-language" variance and the "inter-language" correlation. They differ in the choice of the underlying distributions and how they precisely define and combine these two criteria. The following subsections will describe them in more details.

As already emphasized, CLTM mainly relies on defining appropriate similarities between textual objects expressed in different languages. Numerous categorization, clustering and retrieval algorithms focus on defining efficient and powerful measures of similarity between objects, as strengthened recently by the development of kernel methods for textual information access. We will see that the (linear) statistical algorithms used for performing steps (2) and (3) can most of the time be embedded into one valid (Mercer) kernel, so that we can very easily obtain non-linear variants of these algorithms, just by adopting some standard non-linear kernels.

Cross-Language Semantic Analysis

This amounts to concatenate the vectorial representation of each view of the objects of the parallel collection (typically, objects are aligned sentences), and then to perform standard singular value decomposition of the global object/term matrix. Equivalently, defining the kernel similarity matrix between all pairs of multi-view objects as the sum of the mono-lingual textual similarity matrices, this amounts to perform the eigenvalue decomposition of the corresponding kernel Gram matrix, if a dual formulation is adopted. The number of eigenvalues/eigenvectors that are retained to define the latent factors and the corresponding projections is typically from several hundreds of components to several thousands, still much fewer than the original sizes of the vocabulary. Note that this process does not really control the formation of *interlingua* concepts: nothing



Cross-Lingual Text Mining, Fig. 1 Latent dirichlet allocation of a parallel corpus

prevents the method from extracting factors that are linear combination of terms in one language only.

Cross-Language Latent Dirichlet Allocation

The extraction of *interlingua components* is realised by using LDA to model the set of parallel objects, by imposing the same proportion of components (topics) for all views of the same object. This is represented in Fig. 1.

LDA is performing some form of clustering, with a predefined number of components (K) and with the constraint that the two views of the same object belongs to the clusters with the same membership values. This results in $2.K$ component profiles that are then used for "folding in" (projecting) new documents by launching some form of EM to derive their posterior probabilities to belong to each of the language-independent component. The similarity between two documents written in different languages is obtained by comparing their posterior distribution over these latent classes. Note that this approach could easily integrate supervised topic information and provides a nice framework for semi-supervised *interlingua* concept extraction.

Cross-Language Canonical Correlation Analysis

The Primal Formulation

CCA is a standard statistical method to perform multi-block multivariate analysis, the goal being to find linear combinations of variables for each block (i.e., each language) that are maximally correlated. In other words, CCA is able to enforce the commonality of latent concept formations by extracting maximally correlated projections. Starting from a set of paired views of the same objects (typically, aligned sentences of a parallel corpus) in languages L1 and L2, the algebraic formulation of this optimization problem leads to a generalized eigenvalue problem of size $(n_1 + n_2)$, where n_1 and n_2 are the sizes of the vocabularies in L1 and L2 respectively. For obvious scalability reasons, the dual – or kernel – formulation (of size N, the number of paired objects in the training set) is often preferred.

Kernel Canonical Correlation Analysis

Basically, Kernel Canonical Correlation Analysis amounts to do CCA on some implicit, but more complex feature space and to express the projection coefficients as linear combination of the training paired objects. This results in the dual formulation, which is a generalized eigenvalue/vector problem of size $2N$, that involves only the monolingual kernel gram matrices K_1 and K_2 (matrices of monolingual textual similarities between all pairs of objects in the training set in language L1 and L2 respectively). Note that it is easy to show that the eigenvalues go by pairs: we always have two symmetrical eigenvalues $+\lambda$ and $-\lambda$. This kernel formulation has the advantage to include any text specific prior properties in the kernel (e.g., use of N-gram kernels, word-sequence kernels, and any semantically-smoothed kernel). After extraction of the first k generalized eigenvalues/eigenvectors, the similarity between any pair of test objects in languages L1 and L2 can be computed by using projection matrices composed of extracted eigenvector as well as the (monolingual) kernels of the test objects with the training objects.

Regularization and Partial Least Squares Solution

When the number of training examples (N) is less than n_1 and n_2 (the dimensions of the monolingual feature spaces), the eigenvalue spectrum of the KCCA problem has generally two null eigenvalues (due to data centering), $(N - 1)$ eigenvalues in $+1$ and $(N - 1)$ eigenvalues in -1, so that, as such, the KCCA problem only results in trivial solutions and is useless. When using kernel methods, the case $(N < n_1, n_2)$ is frequent, so that some regularization scheme is needed. One way of realizing this regularization is to resort to finding the directions of maximum covariance (instead of correlation): this can be considered as a partial least squares (PLS) problem, whose formulation is very similar to the CCA problem. Adopting a mixed criterion CCA/PLS (trying to maximize a combination of covariance and correlation between projections) turns out to both avoid over-fitting (or spurious solutions) and to enhance numerical stability.

Approximate Solutions

Both CCA and KCCA suffer from a lack of scalability, due to the fact the complexity of generalized eigenvalue/vector decomposition is $O(N^3)$ for KCCA or $O(\min(n_1, n_2)^3)$ for CCA. As it can be shown that performing a complete KCCA (or KPLS) analysis amounts to do first complete PCA's, and then a linear CCA (or PLS) on the resulting new projections, it is obvious that we could reduce the complexity by working on a reduced-rank approximation (incomplete KPCA) of the kernel matrices. However, the implicit projections derived from incomplete KPCA may be not optimal with respect to cross-correlation or covariance criteria. Another idea to decrease the complexity is to perform some incomplete Cholesky decomposition of the (monolingual) kernel matrices K_1 and $K \quad _2$ (that is equivalent to partial Gram-Schmit orthogonalisation in the feature space): $K_1 = G_1 . G_1^t$ and $K_2 = G_2 . G_2^t$, with G_i of rank $k \ll N$. Considering G_i as the new representation of the training data, KCCA now reduces to solving a generalized eigenvalue problem of size $2.k$.

Specific Applications

The previous sections illustrated a number of different ways of solving the core problem of cross-language text mining: quantifying the similarity between two spans of text in different languages. In this section we turn to describing some actual applications relying on these methods.

Cross-Language Information Retrieval (CLIR)

Given a collection of documents in several languages and a single query, the CLIR problem consists in producing a single ranking of all documents according to their relevance to the query. CLIR is in particular useful whenever a user has some knowledge of the languages in which documents are written, but not enough to express his/her information needs in those languages by means of a precise query. Sometimes CLIR engines are coupled with translation tools to help the user access the content of relevant documents written in languages unknown to him/her. In this case document collections in an even larger number of languages can be effectively queried.

It is probably fair to say that the vast majority of the CLIR systems use a translation-based approach. In most cases it is the query which is translated in all languages before being sent to monolingual search engines. While this limits the amount of translation work that needs be done, it requires doing it on-line at query time. Moreover, when queries are short it can be difficult to translate them correctly, since there is little context to help identifying the correct sense in which words are used. For these reasons several groups also proposed translating all documents at indexing time instead. Regardless of whether queries or documents are translated, whenever similarity scores between (possibly translated) queries and (possibly translated) documents are not directly comparable, all methods then face the problem of merging multiple monolingual rankings in a single multilingual ranking.

Research in CLIR and cross-language question answering (see below) has been significantly stimulated by at least three government-sponsored evaluation campaigns:

- The NII Test Collection for IR Systems (NTCIR) (http://research.nii.ac.jp/ntcir/), running yearly since 1999, focusing on Asian languages (Japanese, Chinese, Korean) and English.
- The Cross-Language Evaluation Forum (CLEF) (http://www.clef-campaign.org), running yearly since 2000, focusing on European languages.
- A cross-language track at the Text Retrieval Conference (TREC) (http://trec.nist.gov/), which was run until 2002, focused on querying documents in Arabic using queries in English.

The respective websites are ideal starting points for any further exploration on the subject.

Cross-Language Question Answering (CLQA)

Question answering is the task of automatically finding the answer to a specific question in a document collection. While in practice this vague description can be instantiated in many different ways, the sense in which the term is mostly understood is strongly influenced by the task specification formulated by the National Institute of Science and Technology (NIST) of the United States for its TREC evaluation conferences (see above). In this sense, the task consists in identifying a *text snippet*, i.e., a substring, of a predefined maximal length (e.g., 50 characters, or 200 characters) within a document in the collection containing the answer. Different classes of questions are considered:

- Questions around facts and events.
- Questions requiring the definition of people, things and organizations.
- Questions requiring as answer lists of people, objects or data.

Most proposals for solving the QA problem proceed by first identifying promising documents (or document segments) by using information retrieval techniques treating the question as a query, and then performing some finer-grained analysis to converge to a sufficiently short snippet. Questions are classified in a hierarchy of possible "question types." Also, documents are preliminarily indexed to identify elements (e.g., person names) that are potential answers to questions of relevant types (e.g., "Who" questions).

Cross-language question answering (CLQA) is the extension of this task to the case where the collection contains documents in a language different than the language of the question. In this task a CLIR step replaces the monolingual IR step to shortlist promising documents. The classification of the question is generally done in the source language.

Both CLEF and NTCIR (see above) organize cross-language question answering comparative evaluations on an annual basis.

Cross-Language Categorization (CLCat) and Clustering (CLCLu)

Cross-language categorization tackles the problem of categorizing documents in different languages in a same categorization scheme.

The vast majority of document categorization systems rely on machine learning techniques to automatically acquire the necessary knowledge (often referred to as a *model*) from a possibly large collection of manually categorized documents. Most often the model is based on frequency counts of words, and is thus intrinsically language-dependent. The most direct way to perform categorization in different languages would consist in manually categorizing a sufficient amount of documents in all languages of interest and then train a set of independent categorizer. In some cases, however, it is impractical to manually categorize a sufficient number of documents to ensure accurate categorization in all languages, while it can be easier to identify bilingual dictionaries or parallel (or comparable) corpora for the language pairs and in the application domain of interest. In such cases it is then preferable to obtain manually categorized documents only for a single language *A* and use them to train a monolingual categorizer. Any of the translation-based approaches described above can then be used to translate a document originally in language B – or most often its representation as a *bag of words*– into language A. Once the document is translated, it can be categorized using the monolingual A system.

As an alternative, latent-semantics approaches can be used as well. An existing parallel corpus can be used to identify an abstract vector space common to A and B. The manually categorized documents in A can then be represented in this space, and a model can be learned which operates directly on this latent-semantic representation. Whenever a document in B needs be categorized, it is first projected in the common semantic space and then categorized using the same model.

All these considerations carry unchanged to the cross-language clustering task, which consists in identifying subsets of documents in a multilingual document collection which are mutually similar to one another according to some criterion. Again, this task can be effectively solved by either translating all documents into a single language or by learning a common semantic space and performing the clustering task there.

While CLCat and Clustering are relevant tasks in many real-world situations, it is probably fair to say that less effort has been devoted to them by the research community than to CLIR and CLQA.

Recommended Reading

Brown PE, Della Pietra VJ, Della Pietra SA, Mercer RL (1993) The mathematics of statistical machine translation: parameter estimation. Comput Linguist 12(2):263–311

Gaussier E, Renders J-M, Matveeva I, Goutte C, Déjean H (2004) A geometric view on bilingual lexicon extraction from comparable corpora. In: Proceedings of the 42nd annual meeting of the association for computational linguistics, Barcelona. Association for Computational Linguistics, Morristown

Savoy J, Berger PY (2005) Report on CLEF-2005 evaluation campaign: monolingual, bilingual and GIRT information retrieval. In: Proceedings of the cross-language evaluation forum (CLEF). Springer, Heidelberg, pp 131–140

Zhang Y, Vines P (2005) Using the web for translation disambiguation. In: Proceedings of the NTCIR-5 workshop meeting, Tokyo

Cross-Validation

Definition

Cross-validation is a process for creating a distribution of pairs of training and ▸ test sets out of a single ▸ data set. In cross validation the data are partitioned into k subsets, $S_1 \ldots S_k$, each called a *fold*. The folds are usually of approximately the same size. The learning algorithm is then applied k times, for $i = 1$ to k, each time using the union of all subsets other than S_i as the ▸ training set and using S_i as the ▸ test set.

Cross-References

▸ Algorithm Evaluation
▸ Leave-One-Out Cross-Validation

Cumulative Learning

Pietro Michelucci[1] and Daniel Oblinger[2]
[1]Strategic Analysis, Inc., Arlington, VA, USA
[2]DARPA/IPTO, Arlington, VA, USA

Synonyms

Continual learning; Lifelong learning; Sequential inductive transfer

Definition

Cumulative learning (CL) exploits knowledge acquired on prior tasks to improve learning performance on subsequent related tasks. Consider, for example, a CL system that is learning to play chess. Here, one might expect the system to learn from prior games concepts (e.g., favorable board positions, standard openings, end games, etc.) that can be used for future learning. This is in contrast to base learning (Vilalta and Drissi 2002) in which a fixed learning algorithm is applied to a single task and performance tends to improve only with more exemplars. So, in CL there tends to be explicit reuse of learned knowledge to constrain new learning, whereas base learning depends entirely upon new external inputs.

Relevant techniques for CL operate over multiple tasks, often at higher levels of abstraction, such as new problem space representations, task-based selection of learning algorithms, dynamic adjustment of learning parameters, and iterative analysis and modification of the learning algorithms themselves. Though actual usage of this term is varied and evolving, CL typically connotes sequential ▶ inductive transfer. It should be noted that the word "inductive" in this connotation qualifies the transfer of knowledge to new tasks, not the underlying learning algorithms.

Related Terminology

The terms "meta-learning" and "learning to learn" are sometimes used interchangeably with CL. However each of these concepts has a specific relationship to CL.

▶ Meta-learning (Brazdil et al. 2009; Vilalta and Drissi 2002) involves the application of learning algorithms to meta-data, which are abstracted representations of input data or learning system knowledge. In the case that abstractions of system knowledge are themselves learning algorithms, meta-learning involves assessing the suitability of these algorithms for previous tasks and, on that basis, selecting algorithms for new tasks (see entry on "▶ Metalearning"). In general, the sharing of abstracted knowledge across tasks in a CL system implies the use of meta-learning techniques. However, the converse is not true. Meta-learning can and does occur in learning systems that do not accumulate and transfer knowledge across tasks.

Learning to learn is a synonym for inductive transfer. Thus, learning to learn is more general than CL. Though it specifies the application of knowledge learned in one domain to another, it does not stipulate whether that knowledge is accumulated and applied sequentially or shared in a parallel learning context.

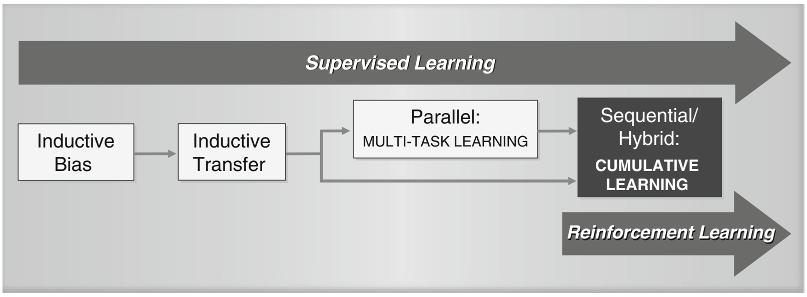
Motivation and Background

Traditional ▶ supervised learning approaches require large datasets and extensive training in order to generalize to new inputs in a single task. Furthermore, traditional (non-CL) ▶ reinforcement learning approaches require tightly constrained environments to ensure a tractable state space. In contrast, humans are able to generalize across tasks in dynamic environments from brief exposure to small datasets. The human advantage seems to derive from the ability to draw upon prior task and context knowledge to constrain hypothesis development for new tasks. Recognition of this disparity between human learning and traditional machine learning had led to the pursuit of methods that seek to emulate the accumulation and exploitation of task-based knowledge that is observed in humans. A coarse evolution of this work is depicted in Fig. 1.

History

Advancements in CL have resulted from two classes of innovation: the development of techniques for ▶ inductive transfer and the integration of those techniques into autonomous learning systems.

Alan Turing (1950) was the first to propose a cumulative learning system. His 1950 paper is best remembered for the imitation game, later known as the Turing test. However, the final sections of the paper address the question of how a machine could be made sufficiently complex to be able to pass the test. He posited that program-



Cumulative Learning, Fig. 1 Evolution of cumulative learning

ming it would be too difficult a task. Therefore, it should be instructed as one might teach a child, starting with simple concepts and working up to more complex ones.

Banerji (1964) introduced the use of predicate logic as a description language for machine learning. Thus, Banerji was one of the earliest advocates of what would later become ► ILP. His concept description language allowed the use of background knowledge and therefore was an extensible language. The first implementation of a cumulative learning system based on Banerjis ideas was Cohens CONFUCIUS (Cohen 1978; Cohen and Sammut 1982). In this work, an instructor teaches the system concepts that are stored in a long-term memory. When examples of a new concept are seen, their descriptions are matched against stored concepts, which allow the system to re-describe the examples in terms of the background knowledge. Thus, as more concepts are accumulated, the system is capable of describing complex objects more compactly than if it had not had the background knowledge. Compact representations generally allow complex concepts to be learned more efficiently. In many cases, learning would be intractable without the prior knowledge. See the entries on ► Inductive Logic Programming, which describe the use of background knowledge further.

Independent of the research in symbolic learning, much of the ► inductive transfer research that underlies CL took root in ► artificial neural network research, a traditional approach to ► supervised learning. For example, Abu-Mostafa (1990) introduced the notion of reducing the hypothesis space of a neural network by introducing "hints" either as hard-wired additions to the network or via examples designed to teach a particular invariance. The task of a neural network can be thought of as the determination of a function that maps exemplars into a classification space. So, in this context, hints constitute an articulation of some aspect of the target mapping function. For example, if a neural network is tasked with mapping numbers into primes and composites, one "hint" would be that all even numbers (besides 2) are composite. Leveraging such a priori knowledge about the mapping function may facilitate convergence on a solution. An inherent limitation to neural networks, however, is their immutable architecture, which does not lend itself to the continual accumulation of knowledge. Consequently, Ring (1991) introduced a neural network that constructs new nodes on demand in a reinforcement learning context in order to support ongoing hierarchical knowledge acquisition and transfer. In this model, nodes called "bions" correspond simultaneously to the enactment and perception of a single behavior. If two bions are activated in sequence repeatedly, a new bion is created to join the coincident pair and represent their collective functionality.

Contemporaneously, Pratt et al. (1991) investigated the hypothesis that knowledge acquired by one neural network could be used to assist another neural network learn a related task. In the speech recognition domain, they trained three separate networks, each corresponding to speech segments of a different length, such that each network was optimized to learn certain types of phonemes. They then demonstrated that a direct transfer of information encoded as network weights from these three specialized networks to a single, combined speech recognition network resulted in a tenfold reduction in training epochs for the combined network compared with the number of training epochs required when no knowledge was transferred. This was one of the first empirical results in neural network-based transfer learning. Caruana (1993) extended this work to demonstrate the performance benefits associated with the simultaneous transfer of ▶ inductive bias in a "Multitask Learning" (MTL) methodology. In this work, Caruana hypothesized that training the same neural network simultaneously on related tasks would naturally induce additional constraints on learning for each individual task. The intuition was that converging on a mapping in support of multiple tasks with shared representations might best reveal aspects of the input that are invariant across tasks, thus obviating within-task regularities, which might be less relevant to classification. Those empirical results are supported by Baxter (1995) who proved that the number of examples required by a representation learner for learning a single task is an inverse linear function of the number of simultaneous tasks being learned.

Though the innovative underpinnings of inductive transfer that critically underlie CL evolved in a supervised learning context, it was the integration of those methods with classical reinforcement learning that has led to current models of CL. Early integration of this type comes from Thrun and Mitchell (1995), who applied an extension of explanation-based learning (EBL), called explanation-based neural networks (EBNN) (Mitchell and Thrun 1993), to an agent-based "lifelong learning framework." This framework provides for the acquisition of different control policies for different environments and reward functions. Since the robot actuators, sensors, and the environment (largely) remain invariant, this framework supports the use of knowledge acquired from one control problem to be applied to another. By using EBNN to allow learning from previous control problems to constrain learning on new control problems, learning is accelerated over the lifetime of the robot.

More recently, Silver and Mercer (2002) introduced a hybrid model that involves a combination of parallel and sequential inductive transfer in an autonomous agent framework. The so-called task rehearsal method (TRM) uses MTL to combine new training inputs with relevant exemplars that are generated from prior task knowledge. Thus, inductive bias is achieved by training the neural networks on new tasks while simultaneously rehearsing learned task knowledge.

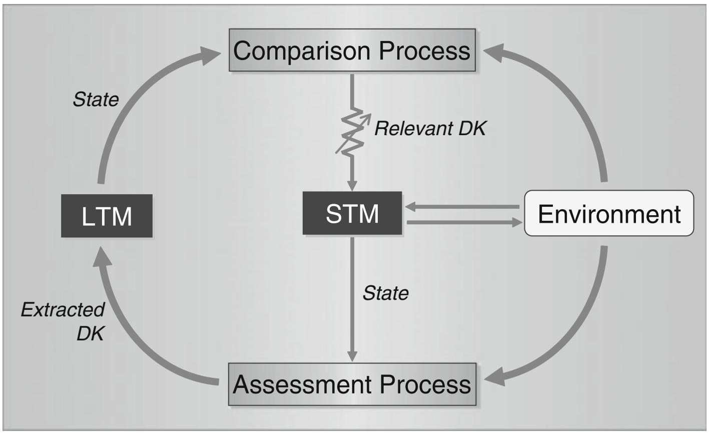
Structure of the Learning System

CL is characterized by systems that use prior knowledge to bias future learning. The canonical interpretation is that knowledge transfer occurs at the task level. Although this description encompasses a broad research space, it is not boundless. In particular, CL systems must be able to (1) retain knowledge and (2) use that knowledge to restrict the hypothesis space for new learning. Nonetheless, learning systems can vary widely across numerous orthogonal dimensions and still meet these criteria.

Toward a CL Specification

Recognizing the empirical utility of a more specific delineation of CL systems, Silver and Poirier (2005) introduced a set of functional requirements, classification criteria, and performance specifications that characterize more precisely the scope of machines capable of lifelong learning. Any system that meets these requirements is considered a machine lifelong learning (ML3) system. A general CL architecture that conforms to the ML3 standard is depicted in Fig. 2.

Two basic memory constructs are typical of CL systems. Long term memory (LTM) is required for storing domain knowledge (DK) that can be used to bias new learning. Short term memory (STM) provides a working memory for building representations and testing hypotheses associated with new task learning. Most of the ML3 requirements specify the interplay of these constructs.



Cumulative Learning, Fig. 2 Typical CL system

LTM and STM are depicted in Fig. 2, along with a comparison process, an assessment process, and the learning environment. In this model, the comparison process evaluates the training input in the context of LTM to determine the most relevant domain knowledge that can be used to constrain short term learning. The comparison process also determines the weight assigned to domain knowledge that is used to bias short term learning. Once the rate of performance improvement on the primary task falls below a threshold the assessment process compares the state of STM to the environment to determine which domain knowledge to extract and store in LTM.

Classification of CL Systems

The simplicity of the architecture shown in Fig. 2 belies the richness of the feature space for CL systems. The following classification dimensions are derived largely from the ML3 specification. This list includes both qualitative and quantitative dimensions. They are presented in three overlapping categories: architectural features, characteristics of the knowledge base, and learning capabilities.

Architecture

The following architectural dimensions for a CL system range from paradigm choices to low-level interface considerations.

Learning paradigm – The learning paradigm(s) may include supervised learning (e.g., neural network, SVM, ILP, etc.), unsupervised learning (e.g., clustering), reinforcement learning (e.g., automated agent), or some combination thereof. Figure 2 depicts a general architecture with processes that are common across these learning paradigms, and which could be elaborated to reflect the details of each.

Task order – CL systems may learn tasks sequentially (Thrun and Mitchell 1995), in parallel (e.g., Caruana 1993), or via a hybrid methodology (e.g., TRM Silver and Mercer 2002). One hybrid approach is to engage in practice (i.e., revisiting prior learned tasks). Transferring knowledge between learned tasks through practice may serve to improve generalization accuracy. Task order would be reflected in the sequence of events within and among process arrows in the Fig. 2 architecture. For example, a system may alternate between processing new exemplars and "practicing" with old, stored exemplars.

Transfer method – Knowledge transfer can also be representational or functional. Functional transfer provides implicit pressure from related training exemplars. For example, the environmental input in Fig. 2 may take the form of training exemplars drawn randomly from data representing two related tasks, such that learning to classify exemplars from one task implicitly constrains learning on the other task. Representational knowledge transfer involves the direct or indirect (Pratt et al. 1991) assignment of a hypothesis representation. A direct inductive transfer entails the assignment of an original hypothesis representation, such as a vector of trained neural network activation weights. This might take the form of a direct injection to LTM in Fig. 2. Indirect transfer implies that some level of abstraction analysis has been applied to the hypothesis representation prior to assignment.

Learning stages – A learning system may implement learning in a single stage or in a series of stages. An example of a two-stage system is one that waits to initiate the long-term storage of domain knowledge until after primary task learning in short-term memory is complete. Like task order, learning stages would be reflected in the sequence of events within and among process arrows in the Fig. 2 architecture. But in this case, ordering pertains to the manner in which learning is staged across encoding processes.

Interface cardinality – The interface cardinality can be fixed or variable. Fixing the number of inputs and outputs has the advantage of providing a consistent interface without posing restrictions on the growth of the internal representation.

Data type – The input and output data types can be fixed or variable. A type-flexible system can produce both categorical and scalar predictions.

Scalability – CL systems may or may not scale on a variety of dimensions including inputs, outputs, training examples, and tasks.

Knowledge

This category pertains to the long-term storage of learned knowledge. Thus, the following CL dimensions characterize knowledge representation, storage, and retrieval.

Knowledge representation – Stored knowledge can manifest as functional or representational. Functional knowledge retention involves the storage of specific exemplars or parameter values, which tends to be more accurate, whereas representational knowledge retention involves the storage of hypotheses derived from training on exemplars, which has the advantage of storage economy.

Retention efficacy – The efficacy of long term retention varies across CL systems. Effective retention implies that only domain knowledge with an acceptable level of accuracy is retained so that errors aren't propagated to future hypotheses. A related consideration is whether or not the consolidation of new domain knowledge degrades the accuracy of current or prior hypotheses.

Retention efficiency – The retention efficiency of long term memory can vary according to both economy of representation and computationally efficiency.

Indexing method – The input to the comparison process used to select appropriate knowledge for biasing new learning may simply be exemplars (as provided by LTM in Fig. 2) or may take a representational form (e.g., a vector of neural network weights).

Indexing efficiency – CL systems vary in terms of the speed and accuracy with which they can identify related prior knowledge that is suitable for inductive transfer during short term learning. The input to this selection process is the indexing method.

Meta-knowledge – CL systems differentially exhibit the ability to abstract, store, and utilize meta-knowledge, such as characteristics of the input space, learning system parameter values, etc.

Learning

While all of the dimensions listed herein impact learning, the following dimensions correspond to specific learning capabilities or learning performance metrics.

Agency – The agency of a learning system is the degree of sophistication exhibited by its top-level controller. For example a learning system may be on the low end of the agency continuum if it always applies one predetermined learning method to one task or on the high end if it selects among many learning methods as a function of the learning task. One might imagine, for example, two process diagrams such as the one depicted in Fig. 2, that share the same LTM, but are otherwise distinct and differentially activated by a governing controller as a function of qualitative aspects of the input.

Utility – Domain knowledge acquisition can be deliberative in the sense that the learning system decides which hypotheses to incorporate based upon their estimated utility, or reflexive, in which case all hypotheses are stored irrespective of utility considerations.

Task awareness – Task awareness characterizes the system's ability to identify the beginning and end of a new task.

Bias modulation – A CL system may have the ability to determine the extent to which short-term learning would benefit from inductive transfer and, on that basis, assign a relevant weight. The depth of this analysis can vary and might consider factors such as the estimated sample complexity, number of exemplars, the generalization accuracy of retained knowledge, and relatedness of retained knowledge.

Learning efficacy – A measure of learning efficacy is derived by comparing generalization performance in the presence and absence of an inductive bias. Learning is considered effective when the application of an inductive bias results in greater generalization performance on the primary task than when the bias is absent.

Learning efficiency – Similarly, learning efficiency is assessed by comparing the computational time needed to generate a hypothesis in the presence and absence of an inductive bias. Lower computational time in the presence of bias signifies greater learning efficiency.

The Research Space

Table 1 summarizes the classification dimensions, providing an overview of the research space, an evaluative framework for assessing and contrasting CL approaches, and a generative framework for identifying new areas of exploration. In addition, checked items in the Values column indicate ML3 guidance. Specifically, an ideal ML3 system would correspond functionally to the called-out items and performance criteria. However, Silver and Poirier (2005) allude to the fact that it would be nigh impossible to generate a strictly compliant ML3 system since some of the recommended criteria do not coexist easily. For example, effective and efficient learning are mutually incompatible because they require different forms of knowledge transfer. Nonetheless, a CL system that falls within scope of the majority of the ML3 criteria would be well-positioned to exhibit lifelong learning behavior.

Future Directions

Emergent work (Oblinger 2006; Swarup et al. 2006) in instructable computing has given rise to a new CL paradigm that is largely ML3 compliant and involves high degrees of task awareness and agency sophistication. Swarup et al. (2006) describe an approach in which domain knowledge is represented in the form of structured graphs. Short term (primary task) learning occurs via a genetic algorithm, after which domain knowledge is extracted by mining frequent subgraphs. The accumulated domain knowledge forms an ontology to which the learning system grounds symbols as a result of structured interactions with instructional agents. Subsequent interactions occur using the symbol system as a shared lexicon for communication between the instructor and the learning system. Knowledge acquired from these interactions bootstrap future learning.

The Bootstrapped Learning framework proposed by Oblinger (2006) provides for hierarchical, domain-independent learning that, like the effort described above, is also premised on a model of building concepts from structured lessons. In this case, however, there is no a priori knowledge acquisition. Instead, some "common" knowledge about the world is provided explicitly to the learning system, and then lessons are taught by a human teacher using the same natural instruction methods that would be used to teach another human. Rather than requiring a specific learning algorithm, this framework provides a context for evaluating and comparing learning algorithms. It includes a knowledge representation language that supports syntactic, logical, procedural, and functional knowledge, an interaction language for communication among the learning system, instructor, and environment, and an integration architecture that evaluates, processes, and responds to interaction language communiqués in the context of existing knowledge and through the selective utilization of available learning algorithms.

Cumulative Learning, Table 1 CL system dimensions

Category	Dimension	Values (ML3 guidance is indicated by {✓})
Architecture	Learning paradigm	Supervised learning
		Reinforcement learning
		Unsupervised learning
		{✓} Hybrid
	Task order	Sequential
		Parallel
		{✓} Revisit (practice)
		Hybrid
	Transfer method	Functional
		Representational – direct
		Representational – indirect
	Learning stages	{✓} Single (computational retention efficiency)
		Multiple
	Interface cardinality	{✓} Fixed
		Variable
	Data type	Fixed
		Variable
	Scalability	{✓} Inputs
		{✓} Outputs
		{✓} Exemplars
		{✓} Tasks
Knowledge	Representation	Functional
		Representational – disjoint
		{✓} Representational – continuous
	Retention efficacy	{✓} Improves prior task performance
		{✓} Improves new task performance
	Retention efficiency	{✓} Space (memory usage)
		{✓} Time (computational processing)
	Indexing method	{✓} Deliberative – functional
		{✓} Deliberative – representational
		Reflexive
	Indexing efficiency	{✓} Time $< \mathrm{O}(n^c), c > 1(n = \text{tasks})$
	Meta-knowledge	{✓} Probability distribution of input space
		Learning curve
		Error rate
Learning	Agency	Single learning method
		Task-based selection of learning method
	Utility	Single learning method
		Task-based selection of learning method
	Task awareness	Task boundary identification (begin/end)
	Bias modulation	{✓} Estimated sample complexity
		{✓} Number of task exemplars
		{✓} Generalization accuracy of retained knowledge
		{✓} Relatedness of retained knowledge
	Learning efficacy	{✓} Generalization $\mid$ bias $\geq$ generalization $\mid$ no bias
	Learning efficiency	{✓} Time $\mid$ bias $\leq$ time $\mid$ no bias

The learning performance advantages anticipated by these proposals for instructional computing seem to stem from the economy of representation afforded by hierarchical knowledge combined with the tremendous learning bias imposed by explicit instruction.

Recommended Reading

Abu-Mostafa Y (1990) Learning from hints in neural networks (invited). J Complex 6(2):192–198

Banerji RB (1964) A language for the description of concepts. General Syst 9:135–141

Baxter J (1995) Learning internal representations. In: (COLT): proceeding of the workshop on computational learning theory, Santa Cruz. Morgan Kaufmann

Brazdil P, Giraud-Carrier C, Soares C, Vilalta R (2009) Metalearning applications to data mining. Springer

Caruana R (1993) Multitask learning: a knowledge-based source of inductive bias. In: Proceedings of the tenth international conference on machine learning, University of Massachusetts, Amherst, pp 41–48

Caruana R (1996) Algorithms and applications for multitask learning. In: Machine learning: proceedings of the 13th international conference on machine learning (ICML 1996), Bari. Morgan Kauffmann, pp 87–95

Cohen BL (1978) A theory of structural concept formation and pattern recognition. Ph.D. thesis, Department of Computer Science, The University of New South Wales

Cohen BL, Sammut CA (1982) Object recognition and concept learning with CONFUCIUS. Pattern Recogn J 15(4):309–316

Mitchell T (1980) The need for biases in learning generalizations. Rutgers TR CBM-TR-117

Mitchell TM, Thrun SB (1993) Explanation-based neural network learning for robot control. In: Hanson CG (eds) Advances in neural information processing systems, vol 5. Morgan-Kaufmann, San Francisco, pp 287–294

Nilsson NJ (1996) Introduction to machine learning: an early draft of a proposed textbook, p 12. Online at http://ai.stanford.edu/~nilsson/MLBOOK.pdf. Accessed on 22 July 2010

Oblinger D (2006) Bootstrapped learning proposer information pamphletfor broad agency announcement 07-04. Online at http://fs1.fbo.gov/EPSData/ODA/Synopses/4965/BAA07-04/BLPIPfinal.pdf

Pratt LY, Mostow J, Kamm CA (1991) Direct transfer of learned information among neural networks. In: Proceedings of the ninth national conference on artificial intelligence (AAAI-91), Anaheim, pp 584–589

Ring M (1991) Incremental development of complex behaviors through automatic construction of sensory-motor hierarchies. In: Proceedings of the eighth international workshop (ML91), San Mateo

Silver D, Mercer R (2002) The task rehearsal method of life-long learning: overcoming impoverished data. In: Cohen R, Spencer B (eds) Advances in artificial intelligence, 15th conference of the Canadian society for computational studies of intelligence (AI 2002), Calgary, 27–29 May 2002. Lecture notes in computer science, vol 2338. Springer, London, pp 90–101

Silver D, Poirier R (2005) Requirements for machine lifelong learning. JSOCS technical report TR-2005-009, Acadia University

Swarup S, Lakkaraju K, Ray SR, Gasser L (2006) Symbol grounding through cumulative learning. In: Vogt P et al (eds) Symbol grounding and beyond: proceedings of the third international workshop on the emergence and evolution of linguistic communication, Rome. Springer, Berlin, pp 180–191

Swarup S, Mahmud MMH, Lakkaraju K, Ray SR (2005) Cumulative learning: towards designing cognitive architectures for artificial agents that have a lifetime. Technical report UIUCDCS-R-2005-2514

Thrun S (1998) Lifelong learning algorithms. In: Thrun S, Pratt LY (eds) Learning to learn. Kluwer Academic, Norwell

Thrun S, Mitchell T (1995) Lifelong robot learning. Robot Auton Syst 15:25–46

Turing AM (1950) Computing machinery and intelligence. Mind Mind 59(236):433–460

Vilalta R, Drissi Y (2002) A perspective view and survey of meta-learning. Artif Intell Rev 18:77–95

Curse of Dimensionality

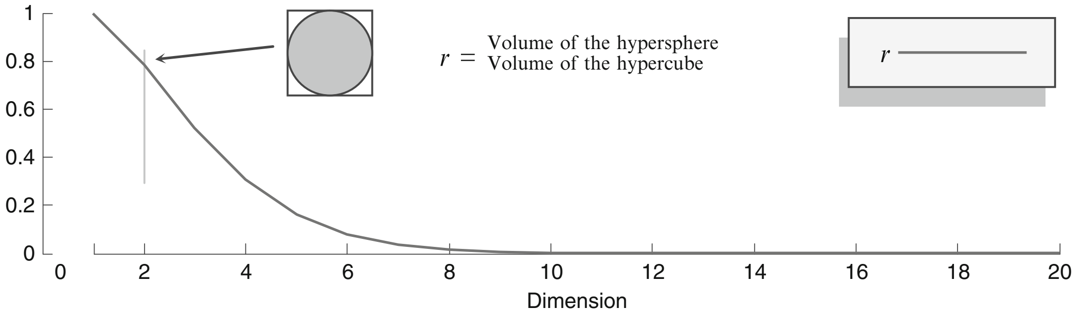
Eamonn Keogh and Abdullah Mueen
University of California-Riverside, Riverside, CA, USA

Definition

The curse of dimensionality is a term introduced by Bellman to describe the problem caused by the exponential increase in volume associated with adding extra dimensions to Euclidean space (Bellman 1957).

For example, 100 evenly-spaced sample points suffice to sample a unit interval with no more than 0.01 distance between points; an equivalent sampling of a 10-dimensional unit hypercube with a grid with a spacing of 0.01 between adjacent points would require 10^{20} sample points: thus, in some sense, the 10D hypercube can be said to be a factor of 10^{18} “larger” than the unit interval.

Informally, the phrase *curse of dimensionality* is often used to simply refer to the fact that one’s intuitions about how data structures, sim-



Curse of Dimensionality, Fig. 1 The ratio of the volume of the hypersphere enclosed by the unit hypercube. The most intuitive example, the unit square and unit circle, are shown as an *inset*. Note that the volume of the hypersphere quickly becomes irrelevant for higher dimensionality

ilarity measures, and algorithms behave in low dimensions do typically generalize well to higher dimensions.

Background

Another way to envisage the vastness of high-dimensional Euclidean space is to compare the size of the unit sphere with the unit cube as the dimension of the space increases: as the dimension increases. As we can see in Fig. 1, the unit sphere becomes an insignificant volume relative to that of the unit cube. In other words, almost all of the high-dimensional space is *far away* from the center.

In research papers, the phrase *curse of dimensionality* is often used as shorthand for one of its many implications for machine learning algorithms. Examples of these implications include:

- ▶ Nearest neighbor searches can be made significantly faster for low-dimensional data by indexing the data with an R-tree, a KD-tree, or a similar spatial access method. However, for high-dimensional data all such methods degrade to the performance of a simple linear scan across the data.
- For machine learning problems, a small increase in dimensionality generally requires a large increase in the numerosity of the data, in order to keep the same level of performance for regression, clustering, etc.
- In high-dimensional spaces, the normally intuitive concept of proximity or similarity may not be qualitatively meaningful. This is because the ratio of an object's nearest neighbor over its farthest neighbor approaches one for high-dimensional spaces (Aggarwal 2001). In other words, all objects are approximately equidistant from each other.

There are many ways to attempt to mitigate the *curse of dimensionality*, including ▶ feature selection and ▶ dimensionality reduction. However, there is no single solution to the many difficulties caused by the effect.

Recommended Reading

Aggarwal CC, Hinneburg A, Keim DA (2001) On the surprising behavior of distance metrics in high dimensional spaces. In: ICDT, London, pp 420–434

Bellman RE (1957) Dynamic programming. Princeton University Press, Princeton

Ding H, Trajcevski G, Scheuermann P, Wang X, Keogh E (2008) Querying and mining of time series data: experimental comparison of representations and distance measures. In: Proceedings of the VLDB endowment, Auckland, vol 1, pp 1542–1552

The major database (SIGMOD, VLDB, PODS), data mining (SIGKDD, ICDM, SDM), and machine learning (ICML, NIPS) conferences typically feature several papers which explicitly address the *curse of dimensionality* each year

D

Data Augmentation

▶ Data Enrichment

Data Cleaning

▶ Data Cleansing

Data Cleansing

Synonyms

Data cleaning; Data reconciliation; Data scrubbing

Data cleansing is the process of detecting and correcting (or removing) corrupt or inaccurate records from data.

Cross-References

▶ Data Preparation

Data Enrichment

Synonyms

Data augmentation; Data integration

Data enrichment is the process of adding to an existing data collection. This commonly involves sourcing of additional information about the data points on which data are already held.

Cross-References

▶ Data Preparation

Data Integration

▶ Data Enrichment

Data Linkage

▶ Record Linkage

Data Matching

▶ Record Linkage

Data mining on Text

▶ Text Mining

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

Data Preparation

Zahraa S. Abdallah[1], Lan Du[1], and
Geoffrey I. Webb[2]
[1]Faculty of Information Technology, Monash
University, Clayton, Melbourne, VIC, Australia
[2]Faculty of Information Technology, Monash
University, Victoria, Australia

Abstract

Before data can be analyzed, they must be organized into an appropriate form. Data preparation is the process of manipulating and organizing data prior to analysis.

Data preparation is typically an iterative process of manipulating raw data, which is often unstructured and messy, into a more structured and useful form that is ready for further analysis. The whole preparation process consists of a series of major activities (or tasks) including data profiling, cleansing, integration, and transformation.

Synonyms

Data preprocessing; Data wrangling

Motivation and Background

Data are collected for many purposes, not necessarily with machine learning or data mining in mind. Consequently, there is often a need to identify and extract relevant data for the given analytic purpose. Every learning system has specific requirements about how data must be presented for analysis, and hence data must be transformed to fulfill those requirements. Further, the selection of the specific data to be analyzed can greatly affect the models that are learned. For these reasons, data preparation is a critical part of any machine learning exercise and is often the most time-consuming part of any nontrivial machine learning or data mining project.

In most cases, the preparation process consists of dozens of transformations and needs to be repeated several times. Despite advances in technologies for working with data, each of those transformations may involve much-handcrafted work and can consume a significant amount of time and effort. Thus, working with huge and diverse data remains a challenge. It is often agreed that data wrangling/preparation is the most tedious and time-consuming aspect of data analysis. It has become a big bottleneck or “iceberg” for performing advanced data analysis, particularly on big data. A recent article in the New York Times For Big-Data Scientists reported that the whole process of data wrangling could account up to 80 % of the time in the analysis cycle. In other words, there is only a small fraction of time for data analysts and scientists to do analysis work. According to the data science report Data science report, published by Crown in 2015, messy and disorganized data are the number one obstacle holding data scientists back. The same study reports that 70 % of a data scientist’s time is spent in cleaning data.

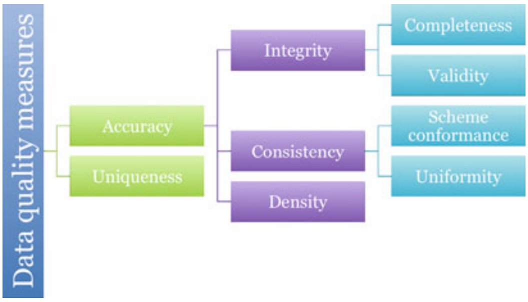
Processes and Techniques

The manner in which data are prepared varies greatly depending upon the analytic objectives for which they are required and the specific learning techniques and software by which they are to be analyzed. The following are a number of key processes and techniques.

Data Profiling: Sourcing, Selecting, and Auditing Appropriate Data

It is necessary to review the data that are already available, assess their suitability to the task at hand, and investigate the feasibility of sourcing new data collected specifically for the desired task. It is also important to assess whether there are sufficient data to realistically obtain the desired machine learning outcomes.

Data quality should also be investigated, as data sets are often of low quality. Those responsible for manual data collection may have little commitment to assuring data accuracy and may take shortcuts in data entry. For example, when default values are provided by a system, these tend to be substantially overrepresented



Data Preparation, Fig. 1 Data quality measures (Adapted from Müller and Freytag 2005)

in the collected data. Automated data collection processes might be faulty, resulting in inaccurate or incorrect data. The precision of a measuring instrument may be lower than desirable. Data may be out-of-date and no longer correct.

Assuring and improving data quality are two of the primary reasons for data preprocessing. There are common criteria to measure and evaluate the quality of data, which can be categorized into two main elements, accuracy and uniqueness (Müller and Freytag 2005), as explained in Fig. 1.

Accuracy is described as an aggregated value over the quality criteria: integrity, consistency, and density. Intuitively this describes the extent to which the data are an exact, uniform, and complete representation of the *mini-world*: the aspects of the world that the data describe. We describe each accuracy criterion as follows:

- **Integrity:** An integral data collection contains representations of all the entities in the mini-world and only of those. Integrity requires both completeness and validity.
  - **Completeness:** Complete data give a comprehensive representation of the mini-world and contain no missing values. We achieve completeness within data cleansing by correcting anomalies and not just deleting them. It is also possible that additional data are generated, representing existing entities that are currently unrepresented in the data. A problem with assessing completeness is that you do not know what you do not know. As a result, there are no known gold standard data, which can be used as a reference to measure completeness.
  - **Validity:** Data are valid when there are no constraints violated. There are numerous mechanisms to increase validity including mandatory fields, enforcing unique values, and data schema/structure.
- **Consistency:** This quality concerns syntactic anomalies as well as contradictions. The main challenge concerning data consistency is choosing which data source you trust for reliable agreement among data across different sources.
  - **Schema conformance:** This is especially true for the relational database systems where the adherence of domain formats relies on the user.
  - **Uniformity:** This is directly related to irregularities.
- **Density:** This criterion concerns the quotient of missing values in the data. There still can be nonexistent values or properties that have to be represented by null values having the exact meaning of not being known.

The above three criteria of integrity, consistency, and density collectively represent the accuracy measure.

The other major quality measure that is also crucial to measure data quality is uniqueness. Uniqueness is satisfied when the data do not contain any duplicates.

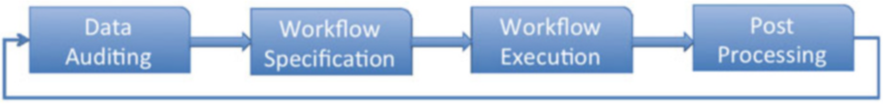
Timeliness is another criterion that also has been considered for data quality. This criterion refers to the currency of the data that keeps it up to date.

More information about data quality can be found in Dasu and Johnson (2003) and Müller and Freytag (2005).

Data Cleansing

Where the data contain noise or anomalies, it may be desirable to identify and remove outliers and other suspect data points or take other remedial action. See ▶ noise.

Data cleansing is defined as the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database. Data cleansing can also be referred to as data cleaning, data scrubbing, or data reconcilia-



Data Preparation, Fig. 2 Data cleansing process (Adapted from Müller and Freytag 2005)

tion. More precisely, the process of data cleansing could be explained as a four-stage process:

1. Define and identify errors in data such as incompleteness, incorrectness, inaccuracy, or irrelevancy.
2. Clean and rectify these errors by replacing, modifying, or deleting them.
3. Document error instances and error types.
4. Measure and verify to see whether the cleansing meets the user's specified tolerance limits in terms of cleanliness.

Data Anomalies

Data are symbolic representations of information, i.e., facts or entities from parts of the world, called a mini-world, depicted by symbolic values. Imperfections in the data set correspond to differences between an ideal (i.e., error-free) data set (DI) and the real data (DR). In this context, anomalousness is a property of data that renders an erroneous representation of the mini-world.

The term *data anomaly* describes any distortion of data resulting from the data collection process. From this perspective, anomalies include duplication, inconsistency, missing values, outliers, noisy data, or any kind of distortion that can cause data imperfections.

Anomalies can be classified at a high level into three categories:

- **Syntactic anomalies:** describe characteristics concerning the format and values used for the representation of the entities. Syntactic anomalies include lexical errors, domain format errors, syntactical errors, and irregularities.
- **Semantic anomalies:** hinder the data collection from being a comprehensive and nonredundant representation of the mini-world. These types of anomalies include integrity constraint violations, contradictions, duplicates, and invalid tuples.
- **Coverage anomalies:** decrease the number of entities and entity properties from the mini-world that is represented in the data collection. Coverage anomalies are categorized as missing values and missing tuples.

Therefore, it is clear that data anomalies can take a number of different forms, each with a different range of analytical consequences.

Data Cleansing Process

Data cleansing is an iterative process that consists of the four consecutive steps (Müller and Freytag 2005), as depicted in Fig. 2:

1. **Data auditing:** This first step mainly identifies the types of anomalies that reduce data quality. Data auditing checks the data using validation rules that are prespecified and then creates a report of the quality of the data and its problems. We often apply some statistical tests in this step for examining the data.
2. **Workflow specification:** The next step is to detect and eliminate anomalies by a sequence of operations on the data. The information collected from data auditing is then used to create a data-cleaning plan. It identifies the causes of the dirty data and plans steps to resolve them.
3. **Workflow execution:** The data-cleaning plan is executed, applying a variety of methods on the data set.
4. **Post-processing and controlling:** The post-processing or control step involves examination of the workflow results and performs exception handling for the data mishandled by the workflow.

Dealing with Missing Values

One major task in data cleansing is dealing with missing values. It is important to determine whether the data have missing values and, if so, to ensure that appropriate measures are taken to allow the learning system to handle this situation See ► missing attribute values.

Handling data that contain missing values is crucial for the data cleansing process and data wrangling in general. In real-life data, most of existing data sets contain missing values that were not introduced or were lost in the recording process for many reasons.

Handling Outliers

An outlier is another type of data anomaly that requires attention in the cleansing process. Outliers are data that do not conform to the overall data distribution.

Outliers can be seen from two different perspectives; first, they might be seen as glitches in the data. Alternatively, they might be also seen as interesting elements that could potentially represent significant elements in the data. For example, outliers in sales records for a store might reflect a successful marketing campaign. Therefore, to classify data as outliers, we must define what the normal behavior of the data is and therefore how different or significant the outlier is relative to normal behavior. There might be different normal behaviors for data and thus different classes of outliers. From the above definition, we can see that as the normality in data differs, various classes of outliers can be detected. To be able to do that, we need to formalize both the normality in the data and inconsistency of the outliers. Read more about handling outliers for data preprocessing in Han et al. (2011).

Data Enrichment/Integration

Existing data may be augmented through data enrichment. This commonly involves sourcing of additional information about the data points on which data are already held. For example, customer data might be enriched by obtaining socioeconomic data about individual customers. The imported data must be integrated with the other data for a unified view of all data sources.

Data integration is a crucial task in data preparation. Combining data from different sources is not trivial especially when dealing with large amounts of data and heterogeneous sources. Data are typically presented in different forms (structured, semi-structured, or unstructured) as well as from different sources (web, database) that could be stored locally or distributed. Moreover, structured data coming from a single source might have different schemas. The combination of these variations is not an easy task.

Integration of data brings many opportunities, yet it also comes with various challenges. We highlight the most relevant challenges below:

1. **Data are heterogeneous:** Data integration involves a combination of data coming from different sources that have been developed independently of each other and thus vary in data format. Each source will have its own schemas, definition of objects, and structure of data (tables, XML, unstructured text, etc.).
2. **The number of sources:** Data integration is already a challenge for a small number of sources, but the challenges are exacerbated when the number of sources grows (such as Web-scale data integration).
3. **Object identity and separate schemas:** Differences exist both on the level of individual objects and the schema level. Every source classifies their data according to taxonomies pertinent to a certain domain.
4. **Time synchronization:** Each source might have a different time window over which data have been captured, different granularities at which events are modeled (daily, weekly, annually), and frequency at which they are updated. Synchronization of these differences and making time-sensitive data compatible are another challenge.
5. **Dealing with legacy data:** There are still important data stored in a legacy form such as IMS, spreadsheets, and ad hoc structures. Combining legacy data with other modern data structures such as XML is a challenging task.
6. **Abstraction levels:** Different data sources might provide data at incompatible levels of abstraction. When combining data, differences in levels of specificity must be resolved.
7. **Data quality:** Data are often erroneous, and combining data often aggravates the problem. Erroneous data has a potentially devastating

impact on the overall quality of the integration process.

The integration process can be divided into two main subtasks, schema integration and data integration, where each has its own techniques and challenges. Schema integration concerns a holistic view across data sources. It focuses on formats, structures, and identification of objects and their level of abstraction. This includes semantic mapping, matching, resolving naming conflicts, and entity resolution. The contents of data add another clue to the integration process.

Even with data from different sources that have identical schemas, integration on the data level is still essential. Data integration deals with different types of problems that concern the data itself rather than the overall structure as in schema integration. Common data integration problems are duplication in data and inconsistency. Correlated or duplicated values/attributes may increase both size and complexity of the data. Resolving conflicts at the data level enhances the overall performance of the integration process.

Data Transformation

It is frequently necessary to transform data from one representation to another. There are many reasons for changing representations:

- **To generate symmetric distributions instead of the original skewed distributions**.
- **Transformation improves visualization** of data that might be tightly clustered relative to a few outliers.
- Data are transformed to achieve **better interpretability.**
- Transformations are often used to **improve the compatibility of the data with assumptions underlying a modeling process**, for example, to linearize (straighten) the relation between two variables whose relationship is nonlinear. Some of the data mining algorithms require the relationship between data to be linear.

In the following, we will discuss different types of transformation whereby each data point $\mathbf{x_i}$ is replaced with a **transformed value** $\mathbf{y_i} = \mathbf{f}(\mathbf{x_i})$, where $\mathbf{f}$ is the transformation function. Many techniques are applied for data transformation. Each technique has its own purpose and dependency on the nature of data. Some of the major transformations are discussed below.

Numeric to Numeric Transformation

Normalization and Rescaling

It is usually the case that raw data are not in a suitable form to be processed by machine learning and data mining techniques. Data normalization is the process of transforming raw data values to another form with properties that are more suitable for modeling and analysis. The normalization process focuses on scaling data in terms of range and distribution. Therefore, it consists of two main processes:

- *Min-max normalization* projects the original range of data onto a new range. Very common normalization intervals are [0,1] and [−1,1]. This normalization method is very useful when we apply a machine learning or data mining approach that utilizes distance. For example, in k-nearest neighbor methods, using un-normalized values might cause attributes whose values have greater magnitudes to dominate over other attributes. Therefore, normalization aims to standardize magnitudes across variables. A useful application for min-max scaling is image processing where pixel intensities have to be normalized to fit within a certain range (i.e., 0–255 for the RGB color range). Also, typical neural network algorithms (ANN) require data that is on a 0–1 scale. Normalization provides the same range of values for each of the inputs to the model.
- **Z-score normalization** (also referred to as standardization) is a normalization method that transforms not only the data magnitude but also the dispersion. Some data mining methods are based on the assumption that data follow a certain distribution. For example, methods such as logistic regression, SVM, and neural network when

using gradient descent/ascent optimization methods assume data follow a Gaussian distribution. Otherwise, the approaches will be ill conditioned and might not guarantee a stable convergence of weight and biases. Other approaches such as linear discriminant analysis (LDA), principal component analysis (PCA), and kernel principal component analysis require features to be on the same scale to find directions that maximize the variance (under the constraints that those directions/eigenvectors/principal components are orthogonal). Z-score normalization overcomes the problem of variables with different units as it transforms variables so that they are centered on 0 with a standard deviation of 1.

- **Decimal scaling** is another type of scaling transformation where the decimal place of a numeric value is shifted so the maximum absolute value will be always less than 1.

Linear Transformation

Linear transformations preserve linear relationships within data. A function f(.) results in a linear transformation if and only if for all values x and y in the original representation, f(x)+f(y) = f(x+y) and f(x)−f(y) = f(x−y). Examples of a linear transformation are transforming Celsius to Fahrenheit, miles to kilometers, and inches to centimeters. All linear transformations follow the standard linear regression formula to convert variables linearly.

Many other transformations are not linear. A nonlinear transformation changes (increases or decreases) linear relationships between variables and, thus, changes the correlation between variables. Examples of nonlinear transformations are square root, raising to a power, logarithm, and any of the trigonometric functions. In the following, we discuss some nonlinear transformation methods.

Power Transformation (Tukey's Ladder of Powers)

Tukey describes a way of re-expressing variables using a power transformation (Tukey 1977). The aim of this transformation is to improve the linearity between variables. When we consider two variables (x and y), transformation can be applied to one variable or both of them depending on the relationship between the two variables. This kind of transformation fits when the relationship between the two variables is monotonic and has a single bend. When the data are represented as pairs of (x,y), Tukey has expressed data transformation as

$$\boldsymbol{y^a = \beta_0 + \beta_1 + x^b}.$$

The choice of $\boldsymbol{a}$ and $\boldsymbol{b}$ decides on the transformation type in the relationship between x and y. Figure 3 shows a visual rule of thumb that has been proposed by John Tukey. The following diagram gives us an insight to understand which transformations are likely to work with different types of data.

We explain Tukey's ladder rule as follows: Suppose the data patterns follow a similar curve as the blue line in Q1; thus the data could be transformed by going up the ladder for x, y, or both. If the data pattern is shaped similar to that shown in Q2, then we should try to transform the data by going the down-ladder for x and/or up-ladder for y. Similar procedures can be applied for the other two quarters. Figure 4 explains the ladder of power for variable y. The transformation is stronger when the power value is away from 1 (the original data) in both directions (up and down).

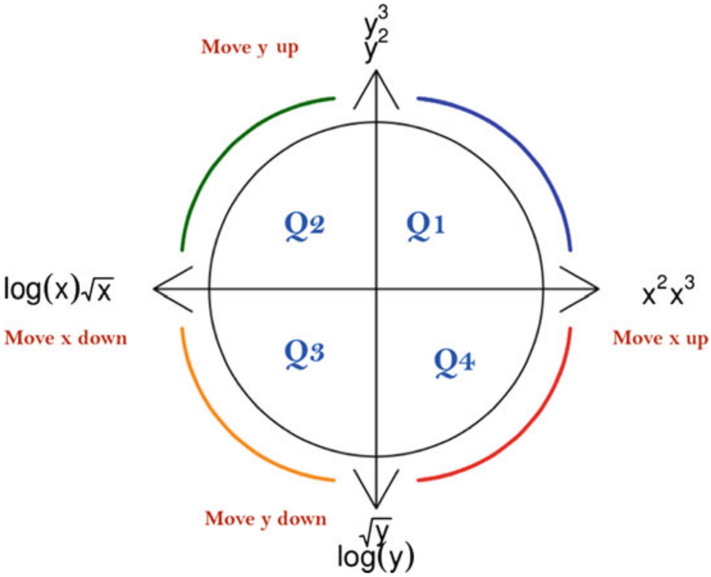
Choosing the Right Numeric Transformation

There is no definite answer to what is the best transformation method to use for a particular data set. The choice is very data dependent and requires an understanding of the domain as well as the data distribution. Trial and error for the common transformation methods may also be required. Table 1 summarizes the main transformation methods.

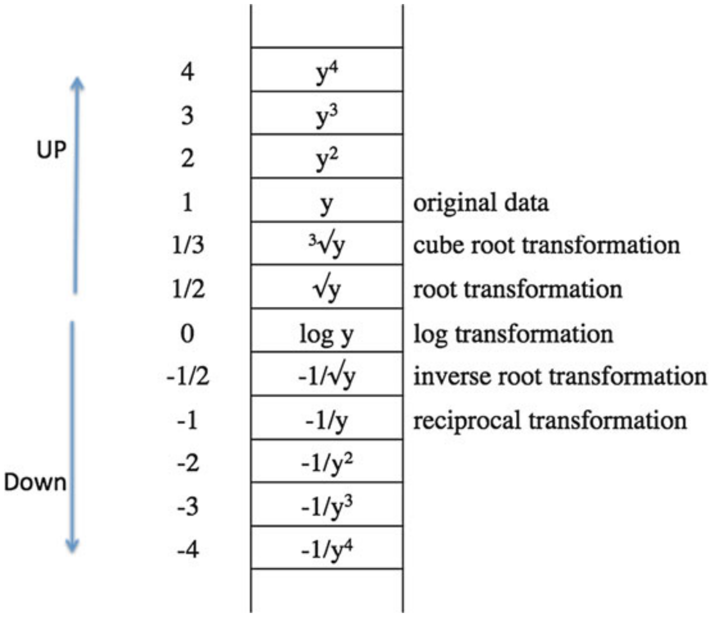
Nominal to Numeric Transformation

All the aforementioned methods transform and re-express numerical variables. However, the transformation of nominal variables is equally

Data Preparation, Fig. 3
Tukey's ladder rule



Data Preparation, Fig. 4
The ladder of power



4	y^4	
3	y^3	
2	y^2	
1	y	original data
1/3	$\sqrt[3]{y}$	cube root transformation
1/2	$\sqrt{y}$	root transformation
0	log y	log transformation
-1/2	$-1/\sqrt{y}$	inverse root transformation
-1	-1/y	reciprocal transformation
-2	$-1/y^2$	
-3	$-1/y^3$	
-4	$-1/y^4$	

important, especially for machine learning and data mining methods that only accept numerical values such as SVM and ANN. Assume that we have a nominal variable x with N different nominal values. There are two approaches to transform x into a numeric variable:

1. The first and simplest approach is to map nominal values to the integers 1 to N. Although simple, this method has two major drawbacks:
   - Integer substitution may impose an ordering that does not actually exist in the original data.
   - The integer value might be used as part of the calculation in the mining algorithm giving an incorrect meaning of weights based on the assigned values.

Data Preparation, Table 1 Summary of key transformation methods

Method	Pros	Cons
Standard linear regression	–Preserves the relationship between variables	–No actual transformation occurred
Reciprocal transformation	–Making small values bigger and big values smaller –Reducing the effect of outliers	–Not applicable for zero
Log transformation	–Good for right skewed data –$\log_{10}(x)$ is especially good at handling higher-order powers of 10 (e.g., 1000,100,000)	–Not applicable for zero and negative values (constant can be added to overcome this)
Root transformation	–Simple counts –Good for right skewed data	–Not applicable for negative values (constant can be added to overcome this)
Logit transformation	–Works with proportions and percents	–Not applicable for 0 and 1 values
Cube root transformation	–Can be applied on negative and 0 values	–Not effective in transformation as long model

2. The other main approach is to first binarize the variable (see 3.7 Binarization) and then map each of the N new binary attributes to the integer values 0 and 1. This approach is generally viewed as safer than the first and hence is more widely used.

Propositionalization

Some data sets contain information expressed in a relational format, describing relationships between objects in the world. While some learning systems can accept relations directly, most operate only on attribute-value representations. Therefore, a relational representation must be re-expressed in attribute-value form. In other words, a representation equivalent to first-order logic must be converted to a representation equivalent only to propositional logic.

Discretization

Discretization transforms continuous data into a discrete form. This is useful in many cases for better data representation, data volume reduction, better data visualization, and representing data at various levels of granularity for data analysis. Data discretization approaches are categorized as supervised, unsupervised, bottom-up, or top-down. Approaches for data discretization include binning, entropy based, nominal to numeric, 3-4-5 rule, and concept hierarchy. See ▶ Discretization.

Binarization

Some systems cannot process multivalued categorical variables. This limitation can be circumvented by binarization, a process that converts a multivalued categorical variable into multiple binary variables, one new variable to represent the presence or absence of each value of the original variable.

Conversely, multiple mutually exclusive binary variables might be converted into a single multivalued categorical variable.

Granularity

It is important to select appropriate levels of granularity for analysis. For example, when distinguishing products, should a gallon of low-fat milk be described as a dairy product, and hence not distinguished from any other type of dairy product; be described as low-fat milk, and hence not distinguished from other brands and quantities; or be uniquely distinguished from all other products?

Analysis at the lowest level of granularity makes possible identification of potentially valuable fine-detail regularities in the data but may make it more difficult to identify high-level relationships.

Dimensionality Reduction

As many learning systems have difficulty with high-dimension data, it may be desirable to project the data onto a lower-dimensional space. Popular approaches to doing so include principal component analysis and kernel methods.

Feature Engineering

It is often desirable to create derived values. For example, the available data might contain fields for purchase price, costs, and sale price. The relevant quantity for analysis might be profit, which must be computed from the raw data.

Feature engineering can be considered as means for dimensionality reduction also, by replacing the original features by a smaller number of derived features.

See ▶ Feature Selection and ▶ Feature Construction in Text Mining.

Sampling

Much of the theory on which learning systems are based assumes that the training data are randomly sampled from the population about which the user wishes to learn a model. However, much historical data contains sampling biases, for example, data that were easy to collect or were considered interesting for some other purpose. It is important to consider whether the available data are sufficiently representative of the future data to which a learned model is to be applied.

In all sampling methods, the aim is to select a sample S containing N instances from the entire data D. Each method models the relationship between a population and a sample with an underlying mathematical process. We discuss in the following some of these methods:

Random Sampling

In this method, S is selected randomly from D with a probability of $1/N$ for any instance in D to be selected. *Simple random sampling* may have very poor performance in the presence of skew in data. There are two main variants of simple random sampling, *with replacement* (SRSWR) and *without replacement* (SRSWOR). For sampling with replacement, the instance that is drawn from the population is replaced, and therefore it might be chosen again. For sampling without replacement, each instance that is drawn from D is removed, and hence S must contain N distinct instances.

Simple random sampling is usually easy to implement and to understand. However, it might cause loss of accuracy if applied to skewed data by failing to include sufficient data to accurately represent the tail of the distribution. The simple random sample might also result in substantial variance across samples.

Cluster Sampling

This method approximates the percentage of each class (or subpopulation of interest) in the overall data set; then it draws a simple random sample from each cluster. In this method, we might not have a complete list of population members (i.e., not all data available). However, a list of groups or "clusters" of this population is available and complete. That means the clusters could be incomplete, but a list of them is complete. Therefore, cluster sampling is a cost-efficient sampling method, as it does not require data to be complete. A drawback for cluster sampling is the possible poor representation of the diversity in clusters.

Stratified Sampling

If D is divided into mutually disjoint parts called strata, obtaining a simple random sample from each stratum generates a stratified sample.

Stratified sampling has a number of advantages. First, inferences can be made about specific subgroups for more efficient statistical estimates. Since each stratum is treated as an independent population, different sampling approaches can be applied to different strata. Second, this method will never result in lower efficiency than the simple random sample, provided that each stratum is proportional to the group's size in the population. Finally, it increases data readability as it represents individual preexisting strata within a population rather than the overall population.

Stratified sampling is complex to implement and estimate. It also can be sensitive to parameters such as selection criteria and minimum group size. Finally, stratified sampling techniques are generally used when the population is heterogeneous, or dissimilar, where certain homogeneous, or similar, subpopulations can be isolated (strata). Thus, this method will not be useful when there are no homogeneous subgroups. Read more about sampling techniques in García et al. (2015).

Balanced sampling is a special case of stratified sampling where the strata correspond to the classes and the sample drawn from each strata is proportional to the class's size in the population.

Cross-References

▸ Anomaly Detection
▸ Binning
▸ Data Set
▸ Dimensionality Reduction
▸ Discretization
▸ Evolutionary Feature Selection and Construction
▸ Feature Construction in Text Mining
▸ Feature Selection
▸ Feature Selection in Text Mining
▸ Kernel Methods
▸ Measurement Scales
▸ Missing Values
▸ Noise
▸ Principal Component Analysis
▸ Propositionalization
▸ Record Linkage

Recommended Reading

Barnett V, Lewis T (1994) Outliers in statistical data, 3rd edn. Wiley series in probability and mathematical statistics. Applied probability and statistics. Wiley, Chichester/New York

Crown (2015) Data science report. http://visit.crowdflower.com/2015-data-scientist-report.html

Dasu T, Johnson T (2003) Exploratory data mining and data cleaning, vol 479. Wiley, New York

Data science report (2014) http://visit.crowdflower.com/2015-data-scientist-report.html

Doan A, Halevy A, Ives Z (2012) Principles of data integration. Morgan Kaufmann, Waltham

For Big-Data Scientists (2014) 'Janitor Work' Is Key Hurdle to Insights. http://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html?r=0&module=ArrowsNav&contentCollection=Technology&action=keypress&region=FixedLeft&pgtype=article (The NYT article by Steve Lohr)

García S, Luengo J, Herrera F (2015) Data preprocessing in data mining. Springer, Cham

Han J, Pei J, Kamber M (2011) Data mining: concepts and techniques. Elsevier, Burlington

Müller H, Freytag J-C (2005) Problems, methods, and challenges in comprehensive data cleansing. Professoren des Inst. Für Informatik, Berlin

Pyle D (1999) Data preparation for data mining. Morgan Kaufmann, San Francisco

Tukey JW (1977) Exploratory data analysis, pp 2–3

Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, Amsterdam/Boston

Data Preprocessing

▸ Data Preparation

Data Scrubbing

▸ Data Cleansing

Data Reconciliation

▸ Data Cleansing
▸ Record Linkage

Data Set

A data set is a collection of data used for some specific machine learning purpose. A ▸ training set is a data set that is used as input to a learning system, which analyzes it to learn a model. A ▸ test set or ▸ evaluation set is a data set containing data that are used to evaluate the model learned by a learning system. A training set may be divided further into a ▸ growing set and a ▸ pruning set. Where the training set and the test set contain disjoint sets of data, the test set is known as a ▸ holdout set.

Data Wrangling

▸ Data Preparation

DBN

Dynamic Bayesian Network. See ▶ Learning Graphical Models

Decision Epoch

In a ▶ Markov decision process, *decision epochs* are sequences of times at which the decision-maker is required to make a decision. In a discrete time Markov decision process, decision epochs occur at regular, fixed intervals, whereas in a continuous time Markov decision process (or semi-Markov decision process), they may occur at randomly distributed intervals.

Decision List

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt, Darmstadt, Deutschland
Department of Information Technology, University of Leoben, Leoben, Austria

Synonyms

Ordered rule set

Definition

A decision list (also called an ordered rule set) is a collection of individual ▶ classification rules that collectively form a ▶ classifier. In contrast to an unordered ▶ rule set, decision lists have an inherent order, which makes classification quite straightforward. For classifying a new instance, the rules are tried in order, and the class of the first rule that covers the instance is predicted. If no induced rule fires, a *default rule* is invoked, which typically predicts the majority class.

Typically, decision lists are learned with a ▶ covering algorithm, which learns one rule at a time, appends it to the list, and removes all covered examples before learning the next one. Decision lists are popular in ▶ inductive logic programming, because PROLOG programs may be considered to be simple decision lists, where all rules predict the same concept.

A formal definition of decision lists, a comparison of their expressiveness to decision trees and rule sets in disjunctive and conjunctive normal form, as well as theoretical results on the learnability of decision lists can be found in Rivest (1987).

Cross-References

▶ Classification Rule
▶ Decision Lists and Decision Trees
▶ Rule Learning
▶ Rule Set

Recommended Reading

Rivest RL (1987) Learning decision lists. Mach Learn 2:229–246

Decision Lists and Decision Trees

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt, Darmstadt, Deutschland
Department of Information Technology, University of Leoben, Leoben, Austria

Definition

▶ Decision trees and ▶ decision lists are two popular ▶ hypothesis languages, which share quite a few similarities, but also have important differences with respect to expressivity and learnability.

Discussion

The key difference between decision trees and decision lists is that the former may be viewed as unordered ▶ rule sets, where each leaf of the tree corresponds to a single rule with a condition part consisting of the conjunction of all edge labels on the path from the root to this leaf. The hierarchical structure of the tree ensures that the rules in the set are non-overlapping, i.e., each example is covered by exactly one rule. This additional constraint makes classification easier (no conflicts from multiple rules), but may result in more complex rules. For example, it has been shown that decision lists (ordered rule sets) with at most k conditions per rule are strictly *more expressive* than decision trees of depth k Rivest (1987).

This is also reflected in the learning strategies that are typically used for learning these concept classes. Decision trees are traditionally learned with a ▸ divide-and-conquer strategy, which successively divides the example space into non-overlapping regions, whereas the ▸ covering algorithm that is typically used for learning rule sets is also known as ▸ separate-and-conquer Fürnkranz (1990) because it successively removes (separates) examples covered by previously learned rule. For a comparison between the two strategies we refer to Boström (1995).

Moreover, the restriction of decision tree learning algorithms to non-overlapping rules imposes strong constraints on learnable rules. One problem resulting from this constraint is the *replicated subtree problem* Pagallo and Haussler (1990); it often happens that identical subtrees have to be learned at various places in a decision tree, because of the fragmentation of the example space imposed by the restriction to non-overlapping rules. Rule learners do not make such a restriction, and are thus less susceptible to this problem. An extreme example for this problem has been provided by Cendrowska (1987), who showed that the minimal decision tree for the concept x defined as

```
IF A = 3 AND B = 3 THEN
   Class = x
IF C = 3 AND D = 3 THEN
   Class = x
```

has 10 interior nodes and 21 leafs assuming that each attribute A ... D can be instantiated with three different values.

On the other hand, a key advantage of decision tree learning is that not only a single rule is optimized, but that conditions are selected in a way that simultaneously optimizes the example distribution in all successors of a node. Attempts to adopt this property for rule learning have given rise to several hybrid systems, the best known being PART Frank and Witten (1998), which learns a decision list that consists of a list of rules, each one being the single best rule of a separate decision tree. This rule can be efficiently found without learning the full tree, by repeated expansion of its most promising branch. Similarly, pruning algorithms can be used to convert decision trees into sets of non-overlapping rules Quinlan (1987a).

See Also

▸ Covering Algorithm
▸ Decision Tree
▸ Divide-and-Conquer Learning
▸ Rule Learning

Recommended Reading

Henrik Boström. Covering vs. divide-and-conquer for top-down induction of logic programs. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1194–1200, 1995.

Jadzia Cendrowska. PRISM: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27:349–370, 1987.

Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. In J. Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning (ICML-98)*, pages 144–151, Madison, Wisconsin, 1998. Morgan Kaufmann.

Johannes Fürnkranz Separate-and-Conquer Rule Learning. *Artificial Intelligence Review*, 13(1):3–54, 1999.

Giulia Pagallo and David Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99, 1990.

John Ross Quinlan. Generating production rules from decision trees. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 304–307. Morgan Kaufmann, 1987a.

Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2:229–246, 1987.

Decision Rule

A decision rule is an element (piece) of knowledge, usually in the form of a "if-then statement":

if < Condition > then < Action >

If its Condition is satisfied (i.e., matches a fact in the corresponding database of a given problem) then its Action (e.g., classification or decision making) is performed. See also ▶ Markovian Decision Rule.

Decision Stump

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt,
Darmstadt, Deutschland
Department of Information Technology,
University of Leoben, Leoben, Austria

Abstract

A *decision stump* is a ▶ Decision Tree, which uses only a single attribute for splitting. For discrete attributes, this typically means that the tree consists only of a single interior node (i.e., the root has only leaves as successor nodes). If the attribute is numerical, the tree may be more complex.

Discussion

Decision stumps perform surprisingly well on some commonly used benchmark datasets from the UCI repository (Holte 1993), which illustrates that learners with a high ▶ Bias and low ▶ Variance may perform well because they are less prone to ▶ Overfitting. Decision stumps are also often used as weak learners in ▶ Ensemble Methods such as boosting Freund and Schapire (1996).

Cross-References

▶ Bias Variance Decomposition
▶ Decision Tree
▶ Overfitting

Recommended Reading

Freund Y, Schapire RE (1996) Experiments with a new boosting algorithm. In: Saitta L (ed) Proceedings of the 13th international conference on machine learning, Bari. Morgan Kaufmann, pp 148–156

Holte RC (1993) Very simple classification rules perform well on most commonly used datasets. Mach Learn 11:63–91

Decision Threshold

The decision threshold of a binary classifier that outputs scores, such as ▶ decision trees or ▶ naive Bayes, is the value above which scores are interpreted as positive classifications. Decision thresholds can be either fixed if the classifier outputs calibrated scores on a known scale (e.g., 0.5 for a probabilistic classifier), or learned from data if the scores are uncalibrated. See ▶ ROC Analysis.

Decision Tree

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt,
Darmstadt, Deutschland
Department of Information Technology,
University of Leoben, Leoben, Austria

Abstract

The induction of decision trees is one of the oldest and most popular techniques for learning discriminatory models, which has been developed independently in the statistical (Breiman et al. 1984; Kass 1980) and machine learning (Hunt et al. 1966; Quinlan 1983, 1986) communities. A *decision tree* is a tree-structured classification model, which is easy to understand, even by non-expert users, and can be efficiently induced from data. An extensive survey of decision-tree learning can be found in Murthy (1998).

Synonyms

Classification tree

Representation

Figure 1 shows a well-known dataset, in which examples are descriptions of weather conditions (*outlook*, *humidity*, *windy*, *temperature*), and the target concept is whether these conditions are suitable for playing golf or not (Quinlan 1986). On the right, a simple decision tree that can be induced from such data is shown. Classification of a new example starts at the top node – the *root* – and the value of the attribute that corresponds to this tree is considered (*outlook* in the example). Classification then proceeds by moving down the branch that corresponds to a particular value of this attribute, arriving at a new node with a new attribute. This process is repeated until we arrive at a terminal node – a so-called *leaf* – which is not labeled with an attribute but with a value of the target attribute (*play golf?*). For all examples that arrive at the same leaf, the same target value will be predicted. Figure 1 shows leaves as rectangular boxes.

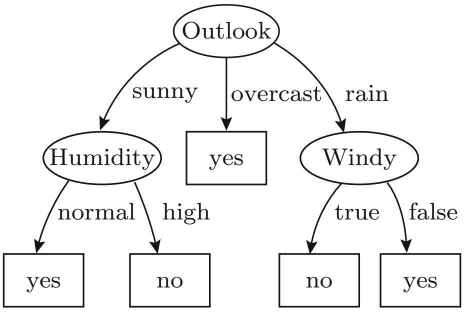
Note that some of the attributes may not occur at all in the tree. For example, the tree in Fig. 1 does not contain a test on *temperature* because the training data can be classified without making a reference to this variable. More generally, one can say that the attributes in the upper parts of the tree (near the root) tend to have a stronger influence on the value of the target variable than the nodes in the lower parts of the tree (e.g., *outlook* will always be tested, whereas *humidity* and *windy* will only be tested under certain conditions).

Learning Algorithm

Decision trees are learned in a top-down fashion, with an algorithm known as *top-down induction of decision trees (TDIDT)*, *recursive partitioning*, or *divide-and-conquer* learning. The algorithm selects the best attribute for the root of the tree, splits the set of examples into disjoint sets, and adds corresponding nodes and branches to the tree. The simplest splitting criterion is for discrete attributes, where each test has the form $t \leftarrow (A = v)$ where v is one possible value of the chosen attribute A. The corresponding set S_t contains all training examples for which the attribute A has the value t. This can be easily adapted to numerical attributes, where one typically uses binary splits of the form $t \leftarrow (A < v_t)$, which indicate whether the attribute's value is above or below a certain threshold value v_t. Alternatively, one can transform the data beforehand using a ▶ discretization algorithm (Fig. 2).

After splitting the dataset according to the selected attribute, the procedure is recursively applied to each of the resulting datasets. If a set contains only examples from the same class, or if no further splitting is possible (e.g., because all possible splits have already been exhausted or all

Outlook	*Temp*	*Humidity*	*Windy*	*Golf?*
rainy	hot	high	false	no
rainy	hot	high	true	no
overcast	hot	high	false	yes
sunny	mild	high	false	yes
sunny	cool	normal	false	yes
sunny	cool	normal	true	no
overcast	cool	normal	true	yes
rainy	mild	high	false	no
rainy	cool	normal	false	yes
sunny	mild	normal	false	yes
rainy	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
sunny	mild	high	true	no



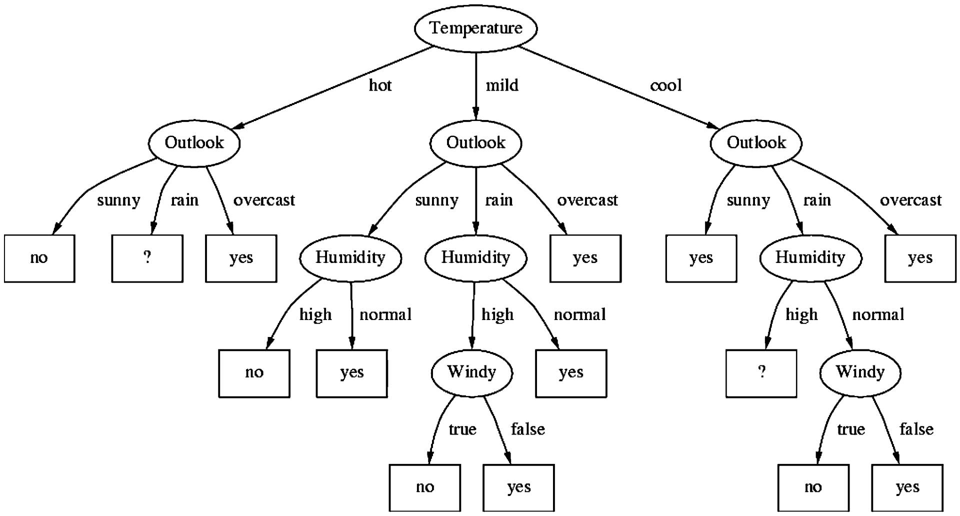
Decision Tree, Fig. 1 A data set describing weather conditions and a target variable (*Play Golf?*) and a decision tree learned for this dataset (Quinlan 1986) remaining splits will have the same outcome for all examples), the corresponding node is turned into a leaf node and labeled with the respective class. For all other sets, an interior node is added and associated with the best splitting attribute for the corresponding set as described above. Hence, the dataset is successively partitioned into non-overlapping, smaller datasets until each set only contains examples of the same class (a so-called *pure* node). Eventually, a pure node can always be found via successive partitions unless the training data contains two identical but contradictory examples, i.e., examples with the same feature values but different class values.

```
function TDIDT($S$)

Input: $S$, a set of labeled examples.

$Tree$ = new empty node
if all examples have the same class $c$
   or no further splitting is possible
then   // new leaf
       LABEL($Tree$) = $c$
else   // new decision node
       $(A, T)$ = FINDBESTSPLIT($S$)
       for each test $t \in T$ do
              $S_t$ = all examples that satisfy $t$
              $Node_t$ = TDIDT($S_t$)
              ADDEDGE($Tree \xrightarrow{t} Node_t$)
       endfor
endif
return $Tree$
```

Decision Tree, Fig. 2 Top-down induction of decision trees

Attribute Selection

The crucial step in decision-tree induction is the choice of an adequate attribute. In the sample tree of Fig. 3, which has been generated from the same 14 training examples as the tree of Fig 1, most leaves contain only single training example, i.e., with the selected splitting criteria, the termination criterion (all examples of a node have to be of the same class) could in many cases only trivially be satisfied (only one example remained in the node). Although both trees classify the training data correctly, the former appears to be more trustworthy, and in practice, one can often observe that simpler trees are more accurate than more complex trees. A possible explanation could be that labels that are based on a higher number of training examples tend to be more reliable. However, this preference for simple models is a heuristic criterion known as ▶ Occam's Razor, which appears to work fairly well in practice. but is still the subject of ardent debates within the machine learning community.



Decision Tree, Fig. 3 A needlessly complex decision tree describing the same dataset

Typical attribute selection criteria use a function that measures the *impurity* of a node, i.e., the degree to which the node contains only examples of a single class. Two well-known impurity measures are the information-theoretic entropy (Quinlan 1986) and the Gini index (Breiman et al. 1984), which are defined as

$$Entropy(S) = -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \cdot \log_2\left(\frac{|S_i|}{|S|}\right)$$

$$Gini(S) = 1 - \sum_{i=1}^{c} \left(\frac{|S_i|}{|S|}\right)^2$$

where S is a set of training examples and S_i is the set of training examples that belong to class c_i. Both functions have their maximum at the point where the classes are equally distributed (i.e., where all S_i have the same size, maximum impurity), and their minimum at the point where one S_i contains all examples ($S_i = S$) and all other S_j, $j \neq i$ are empty (minimum impurity).

A good attribute divides the dataset into subsets that are as pure as possible ideally into sets so that each one only contains examples from the same class. Thus, we want to select the attribute that provides the highest decrease in average impurity, the so-called *gain*:

$$Gain(S, A)$$

$$= Impurity(S) - \sum_{t} \frac{|S_t|}{|S|} \cdot Impurity(S_t)$$

where S_t are non-overlapping disjoint subsets $S_t \in S$ that are induced by splitting the attribute A, and *Impurity* can be any impurity measure. As the first term, *Impurity*(S), is constant for all attributes, one can also omit it and directly minimize the average impurity (which is typically done when *Gini* is used as an impurity measure).

A common problem is that attributes with many values have a higher chance of resulting in pure successor nodes and are therefore often preferred over attributes with fewer values. To counter this, the so-called *gain ratio* normalizes the gained entropy with the intrinsic entropy of the split:

$$GainRatio(S, A) = \frac{Gain(S, A)}{\sum_t \frac{|S_t|}{|S|} \cdot \log_2\left(\frac{|S_t|}{|S|}\right)}$$

A similar phenomenon can be observed for numerical attributes, where the number of possible threshold values determines the number of possible binary splits for this attribute. Numerical attributes with many possible binary splits are often preferred over numerical attributes with fewer splits because they have a higher chance that one of their possible splits fits the data. A discussion of this problem and a proposal for a solution can be found in Quinlan (1996).

Other attribute selection measures, which do not conform to the gain framework laid out above, are also possible, such as CHAID's evaluation with a χ^2 test statistic (Kass 1980). Experimental comparison of different measures can be found in Mingers (1989a) and Buntine and Niblett (1992).

Thus, the final tree is constructed by a sequence of local choices that each consider only those examples that end up at the node that is currently split. Of course, such a procedure can only find local optima for each node, but cannot guarantee convergence to a global optimum (the smallest tree). One of the key advantages of this divide-and-conquer approach is its efficiency, which results from the exponential decrease in the quantity of data to be processed at successive depths in the tree.

Overfitting Avoidance

In principle, a decision-tree model can be fit to any training set that does not contain contradictions (i.e., there are no examples with identical attributes but different class values). This may lead to ▸ Overfitting in the form of overly complex trees.

For this reason, state-of-the-art decision-tree induction techniques employ various ▸ Pruning techniques for restricting the complexity of the found trees. For example, C4.5 has a *pre-pruning* parameter m that is used to prevent further splitting unless at least two successor nodes have at least m examples. The *cost-complexity pruning* method used in CART may be viewed as a simple

▶ Regularization method, where a good choice for the regularization parameter, which trades off the fit of the data with the complexity of the tree, is determined via ▶ Cross-validation.

More typically, *post-pruning* is used for removing branches and nodes from the learned tree. More precisely, this procedure replaces some of the interior nodes of the tree with a new leaf, thereby removing the subtree that was rooted at this node. An empirical comparison of different decision-tree pruning techniques can be found in Mingers (1989b).

It is important to note that the leaf nodes of the new tree are no longer pure nodes, i.e., they no longer need to contain training examples that all belong to the same class. Typically, this is simply resolved by predicting the most frequent class at a leaf. The class distribution of the training examples within the leaf may be used as a reliability criterion for this prediction.

Well-Known Decision-Tree Learning Algorithms

The probably best-known decision-tree learning algorithm is C4.5 (Quinlan 1993), which is based upon (Quinlan 1983), which, in turn, has been derived from an earlier concept learning system (Hunt et al. 1966). ID3 realized the basic recursive partitioning algorithm for an arbitrary number of classes and for discrete attribute values. C4.5 (Quinlan 1993) incorporates several key improvements that were necessary for tackling real-world problems, including handling of numeric and missing attribute values, overfitting avoidance, and improved scalability. A C-implementation of C4.5 is freely available from its author. A re-implementation is available under the name J4.8 in the Weka data mining library. C5.0 is a commercial successor of C4.5, distributed by RuleQuest Research. CART (Breiman et al. 1984) is the best-known system in the statistical learning community. It is integrated into various statistical software packages, such as R or S.

Decision trees are also often used as components in ▶ Ensemble Methods such as random forests (Breiman 2001) or AdaBoost (Freund and Schapire 1996). They can also be modified for predicting numerical target variables, in which case they are known as ▶ regression trees. One can also put more complex prediction models into the leaves of a tree, resulting in ▶ Model Trees.

Cross-References

▶ Decision List
▶ Decision Lists and Decision Trees
▶ Decision Stump
▶ Divide-and-Conquer Learning
▶ Model Trees
▶ Pruning
▶ Regression Trees
▶ Rule Learning

Recommended Reading

Breiman L (2001) Random forests. Mach Learn 45(1): 5–32

Breiman L, Friedman JH, Olshen R, Stone C (1984) Classification and regression trees. Wadsworth & Brooks, Pacific Grove

Buntine W, Niblett T (1992) A further comparison of splitting rules for decision-tree induction. Mach Learn 8:75–85

Freund Y, Schapire RE (1996) Experiments with a new boosting algorithm. In: Saitta L (ed) Proceedings of the 13th international conference on machine learning, Bari. Morgan Kaufmann, pp 148–156

Hunt EB, Marin J, Stone PJ (1966) Experiments in induction. Academic, New York

Kass GV (1980) An exploratory technique for investigating large quantities of categorical data. Appl Stat 29:119–127

Mingers J (1989a) An empirical comparison of selection measures for decision-tree induction. Mach Learn 3:319–342

Mingers J (1989b) An empirical comparison of pruning methods for decision tree induction. Mach Learn 4:227–243

Murthy SK (1998) Automatic construction of decision trees from data: a multi-disciplinary survey. Data Min Knowl Discov 2(4):345–389

Quinlan JR (1983) Learning efficient classification procedures and their application to chess end games. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning. An artificial intelligence approach, Tioga, Palo Alto, pp 463–482

Quinlan JR (1986) Induction of decision trees. Mach Learn 1:81–106

Quinlan JR (1993) C4.5: Programs for machine learning. Morgan Kaufmann, San Mateo

Quinlan JR (1996) Improved use of continuous attributes in C4.5. J Artif Intell Res 4:77–90

Decision Trees for Regression

▸ Regression Trees

Deductive Learning

Synonyms

Analytical learning; Explanation-based learning

Definition

Deductive learning is a subclass of machine learning that studies algorithms for learning provably correct knowledge. Typically such methods are used to speedup problem solvers by adding knowledge to them that is deductively entailed by existing knowledge, but that may result in faster solutions.

Deduplication

▸ Entity Resolution

Deduplication or Duplicate Detection (When Applied to One Database Only)

▸ Record Linkage

Deep Belief Nets

Geoffrey Hinton
University of Toronto, Toronto, ON, Canada

Synonyms

Deep belief networks

Definition

Deep belief nets are probabilistic generative models that are composed of multiple layers of stochastic latent variables (also called "feature detectors" or "hidden units"). The top two layers have undirected, symmetric connections between them and form an associative memory. The lower layers receive top-down, directed connections from the layer above. Deep belief nets have two important computational properties. First, there is an efficient procedure for learning the top-down, generative weights that specify how the variables in one layer determine the probabilities of variables in the layer below. This procedure learns one layer of latent variables at a time. Second, after learning multiple layers, the values of the latent variables in every layer can be inferred by a single, bottom-up pass that starts with an observed data vector in the bottom layer and uses the generative weights in the reverse direction.

Motivation and Background

The perceptual systems of humans and other animals show that high-quality pattern recognition can be achieved by using multiple layers of adaptive nonlinear features, and researchers have been trying to understand how this type of perceptual system could be learned, since the 1950s (Selfridge 1958). Perceptrons (Rosenblatt 1962) were an early attempt to learn a biologically inspired perceptual system, but they did not have an efficient learning procedure for multiple layers of features. Backpropagation (Werbos 1974; Rumelhart et al. 1986) is a supervised learning procedure that became popular in the 1980s because it provided a fairly efficient way of learning multiple layers of nonlinear features by propagating derivatives of the error in the output backward through the multilayer network. Unfortunately, backpropagation has difficulty optimizing the weights in deep networks that contain many layers of hidden units and it requires labeled training data, which is often expensive to obtain. Deep belief nets overcome the limitations of backpropagation by using *unsupervised* learning to create layers of feature detectors that model the statistical structure of the input data without using any information about the required output. High-level feature detectors that capture complicated higher-order statistical structure in the input data can then be used to predict the labels.

Structure of the Learning System

Deep belief nets are learned one layer at a time by treating the values of the latent variables in one layer, when they are being inferred from data, as the data for training the next layer. This efficient, greedy learning can be followed by, or combined with, other learning procedures that fine-tune all of the weights to improve the generative or discriminative performance of the whole network. Discriminative fine-tuning can be performed by adding a final layer of variables that represent the desired outputs and backpropagating error derivatives. When networks with many hidden layers are applied in domains that contain highly structured input vectors, backpropagation learning works much better if the feature detectors in the hidden layers are initialized by learning a deep belief net that models the structure in the input data (Hinton and Salakhutdinov 2006). Matlab code for learning and fine-tuning deep belief nets can be found at http://cs.toronto.edu/~hinton.

Composing Simple Learning Modules

Early deep belief networks could be viewed as a composition of simple learning modules, each of which is a "restricted Boltzmann machine." Restricted Boltzmann machines contain a layer of "visible units" that represent the data and a layer of "hidden units" that learn to represent features that capture higher-order correlations in the data. The two layers are connected by a matrix of symmetrically weighted connections, W, and there are no connections within a layer. Given a vector of activities $\mathbf{v}$ for the visible units, the hidden units are all conditionally independent so it is easy to sample a vector, $\mathbf{h}$, from the posterior distribution over hidden vectors, $p(\mathbf{h}|\mathbf{v}, \mathbf{W})$. It is also easy to sample from $p(\mathbf{v}|\mathbf{h}, \mathbf{W})$. By starting with an observed data vector on the visible units and alternating several times between sampling from $p(\mathbf{h}|\mathbf{v}, \mathbf{W})$ and $p(\mathbf{v}|\mathbf{h}, \mathbf{W})$, it is easy to get a learning signal which is simply the difference between the pairwise correlations of the visible and hidden units at the beginning and end of the sampling (see Chapter ▶ Boltzmann Machines for details).

The Theoretical Justification of the Learning Procedure

The key idea behind deep belief nets is that the weights, $\mathbf{W}$, learned by a restricted Boltzmann machine define both $p(\mathbf{v}|\mathbf{h}, \mathbf{W})$ and the prior distribution over hidden vectors, $p(\mathbf{h}|\mathbf{W})$, so the probability of generating a visible vector, $\mathbf{v}$, can be written as

$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{W})\, p(\mathbf{v}|\mathbf{h}, \mathbf{W}) \qquad (1)$$

After learning $\mathbf{W}$, we keep $p(\mathbf{v}|\mathbf{h}, \mathbf{W})$ but we replace $p(\mathbf{h}|\mathbf{W})$ by a better model of the *aggregated* posterior distribution over hidden vectors – i.e., the nonfactorial distribution produced by averaging the factorial posterior distributions produced by the individual data vectors. The better model is learned by treating the hidden activity vectors produced from the training data as the training data for the next learning module. Hinton et al. (2006) show that this replacement improves a variational lower bound on the probability of the training data under the composite model.

Deep Belief Nets with Other Types of Variable

Deep belief nets typically use the logistic function $y = 1/(1 + \exp(-x))$ of the weighted input, x, received from above or below to determine the probability that a binary latent variable has a value of 1 during top-down generation or bottom-up inference. Other types of variable within the exponential family, such as Gaussian, Poisson, or multinomial, can also be used (Welling et al. 2005; Movellan and Marks 2001) and the variational bound still applies. However, networks with multiple layers of Gaussian or Poisson units are difficult to train and can become unstable. To avoid these problems, the function $\log(1+\exp(x))$ can be used as a smooth approximation to a rectified linear unit. Units of this type often learn features that are easier to interpret than those learned by logistic units. $\log(1+\exp(x))$ is not in the exponential family, but it can be approximated very accurately as a sum of a set of logistic units that all share the same weight vector and adaptive bias term, but differ by having offsets to the shared bias of $-0.5, -1.5, -2.5, \ldots$.

Using Autoencoders as the Learning Module

A closely related approach that is also called a "deep belief net" uses the same type of greedy, layer-by-layer learning with a different kind of learning module – an "autoencoder" that simply tries to reproduce each data vector from the feature activations that it causes (Hinton 1989; Bengio et al. 2007; LeCun and Bengio 2007). However, the variational bound no longer applies, and an autoencoder module is less good at ignoring random noise in its training data (Larochelle et al. 2007).

Applications of Deep Belief Nets

Deep belief nets have been used for generating and recognizing images (Bengio et al. 2007; Hinton et al. 2006; Ranzato et al. 2007), video sequences (Sutskever and Hinton 2007), and motion-capture data (Taylor et al. 2007). If the number of units in the highest layer is small, deep belief nets perform nonlinear dimensionality reduction (Hinton and Salakhutdinov 2006), and by pretraining each layer separately, it is possible to learn very deep autoencoders that can then be fine-tuned with backpropagation (Hinton and Salakhutdinov 2006). Such networks cannot be learned in reasonable time using backpropagation alone. Deep autoencoders learn compact representations of their input vectors that are much better than those found by linear methods such as principal component analysis, and if the highest level code is forced to be binary, they allow extremely fast retrieval of documents or images (Salakhutdinov and Hinton 2007; Torralba et al. 2008).

Recommended Reading

Bengio Y, Lamblin P, Popovici P, Larochelle H (2007) Greedy layer-wise training of deep networks. In: Advances in neural information processing systems, Vancouver, vol 19. MIT, Cambridge

Hinton GE (1989) Connectionist learning procedures. Artif Intell 40(1–3):185–234

Hinton GE, Osindero S, Teh YW (2006) A fast learning algorithm for deep belief nets. Neural Comput 18:1527–1554

Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313:504–507

Larochelle H, Erhan D, Courville A, Bergstra J, Bengio Y (2007) An empirical evaluation of deep architectures on problems with many factors of variation. In: Proceedings of the 24th international conference on machine learning, Corvalis. ACM, New York

LeCun Y, Bengio Y (2007) Scaling learning algorithms towards AI. In: Bottou L et al (eds) Large-scale kernel machines. MIT, Cambridge

Movellan JR, Marks TK (2001) Diffusion networks, product of experts, and factor analysis

Ranzato M, Huang FJ, Boureau Y, LeCun Y (2007) Unsupervised learning of invariant feature hierarchies with applications to object recognition. In: Proceedings of computer vision and pattern recognition conference (CVPR 2007), Minneapolis

Rosenblatt F (1962) Principles of neurodynamics. Spartan Books, Washington, DC

Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323:533–536

Salakhutdinov RR, Hinton GE (2007) Semantic hashing. In: Proceedings of the SIGIR workshop on information retrieval and applications of graphical models, Amsterdam

Selfridge OG (1958) Pandemonium: a paradigm for learning. In: Proceedings of a symposium on mechanisation of though processes, National Physical Laboratory. HMSO, London

Sutskever I, Hinton GE (2007) Learning multilevel distributed representations for high-dimensional sequences. In: Proceedings of the eleventh international conference on artificial intelligence and statistics, San Juan

Taylor GW, Hinton GE, Roweis S (2007) Modeling human motion using binary latent variables. In: Advances in neural information processing systems, Vancouver, vol 19. MIT, Cambridge

Torralba A, Fergus R, Weiss Y (2008) Small codes and large image databases for recognition. In: IEEE conference on computer vision and pattern recognition, Anchorage, pp 1–8
Welling M, Rosen-Zvi M, Hinton GE (2005) Exponential family harmoniums with an application to information retrieval. In: Advances in neural information processing systems, Vancouver, vol 17. MIT, Cambridge, pp 1481–1488
Werbos P (1974) Beyond regression: new tools for prediction and analysis in the behavioral sciences. PhD thesis, Harvard University, Cambridge

Deep Belief Networks

► Deep Belief Nets

Deep Learning

Jürgen Schmidhuber
The Swiss AI Lab, IDSIA, USI & SUPSI,
Manno & Lugano, Switzerland

Abstract

Deep learning artificial neural networks have won numerous contests in pattern recognition and machine learning. They are now widely used by the worlds most valuable public companies. I review the most popular algorithms for feedforward and recurrent networks and their history.

Introduction

Deep learning has revolutionized Pattern Recognition and Machine Learning. It is about credit assignment in adaptive systems with long chains of potentially causal links between actions and consequences.

The ancient term "deep learning" was first introduced to Machine Learning by Dechter (1986) and to artificial neural networks (NNs) by Aizenberg et al. (2000). Subsequently it became especially popular in the context of deep NNs, the most successful deep learners, which are much older though, dating back half a century. This article will focus on essential developments since the 1960s, addressing supervised, unsupervised, and (briefly) reinforcement learning. There is a recent, more detailed survey with 888 references (Schmidhuber 2015). LeCun et al. (2015) provide a more limited view of more recent deep learning history. The present condensed survey is based on the Scholarpedia article (Schmidhuber 2015b).

A standard NN consists of many simple, connected processors called units, each producing a sequence of real-valued activations. Input units get activated through sensors perceiving the environment, other units through connections with real-valued weights from previously active units. Some units may influence the environment by triggering actions. Learning or credit assignment is about finding weights that make the NN exhibit desired behavior, such as controlling a robot. Depending on the problem and how the units are connected, such behavior may require long causal chains of computational stages, where each stage transforms (often in a nonlinear way) the aggregate activation of the network. Deep learning in NNs is about accurately assigning credit across many such stages.

In a sense, sequence-processing recurrent NNs (RNNs) are the ultimate NNs, because they are general computers (an RNN can emulate the circuits of a microchip). In fully connected RNNs, all units have connections to all non-input units. Unlike feedforward NNs, RNNs can implement while loops, recursion, etc. The program of an RNN is its weight matrix. RNNs can learn programs that mix sequential and parallel information processing in a natural and efficient way.

To measure whether credit assignment in a given NN application is of the deep or shallow type, we consider the length of the corresponding credit assignment paths, which are chains of possibly causal connections between subsequent unit activations, e.g., from input units through hidden units to output units in feedforward NNs (FNNs) without feedback connections or through transformations over time in RNNs. FNNs with fixed topology have a problem-independent maximal problem depth bounded by the number of layers of units. RNNs, the deepest of all NNs, may learn to solve problems of potentially unlimited depth, for example, by learning to store in their activation-based "short-term memory" representations of certain important previous observations for arbitrary time intervals.

The difficulty of a problem may have little to do with its depth. Some NNs can quickly learn to solve certain deep but simple problems through random weight guessing (e.g., Hochreiter and Schmidhuber 1997b). In general, however, finding an NN that precisely models a given training set (of input patterns and corresponding labels) is an NP-complete problem and also in the case of deep NNs (e.g., Sima 1994).

First Deep Learners

Certain early NNs (McCulloch and Pitts 1943) did not learn at all. Hebb (1949) published ideas about unsupervised learning. The following decades brought shallow unsupervised NNs and supervised NNs (e.g., Rosenblatt 1958). Early supervised NNs were essentially variants of linear regressors dating back two centuries (Gauss, Legendre).

Deep learning networks originated in the 1960s when Ivakhnenko and Lapa (1965) published the first general, working learning algorithm for supervised deep feedforward multilayer perceptrons. Their units had polynomial activation functions combining additions and multiplications in Kolmogorov-Gabor polynomials. Ivakhnenko (1971) already described a deep network with eight layers trained by the "group method of data handling," still popular in the new millennium. Given a training set of input vectors with corresponding target output vectors, layers are incrementally grown and trained by regression analysis and then pruned with the help of a separate validation set, where regularization is used to weed out superfluous units. The numbers of layers and units per layer can be learned in problem-dependent fashion.

Like later deep NNs, Ivakhnenko's nets learned to create hierarchical, distributed, internal representations of incoming data. Many later nonneural methods of Artificial Intelligence and Machine Learning also learn more and more abstract, hierarchical data representations. For example, syntactic pattern recognition methods (Fu 1977) such as grammar induction discover hierarchies of formal rules to model observations.

Architectures of Convolutional NNs (CNNs)

The 1970s also saw the birth of the convolutional NN (CNN) architecture (Fukushima's Neocognitron, 1979) inspired by neurophysiological insights. Today such architectures are widely used for computer vision. Here the (typically rectangular) receptive field of a unit with given weight vector (a filter) is shifted step by step across a two-dimensional array of input values, such as the pixels of an image (usually there are several such filters). The resulting array of subsequent activation events of this unit can then provide inputs to higher-level units and so on. Due to massive weight replication, relatively few parameters may be necessary to describe the behavior of such convolutional layers, which typically feed downsampling layers consisting of units whose fixed-weight connections originate from physical neighbors in the convolutional layers below. Downsampling units use "spatial averaging" to become active if at least one of their inputs is active; their responses are insensitive to certain small image shifts. Weng (1993) later replaced spatial averaging by "max-pooling" (MP), which is widely used today. Here a two-dimensional layer or array of unit activations is partitioned into smaller rectangular arrays. Each is replaced in a downsampling layer by the activation of its maximally active unit.

Backpropagation

Ivakhnenko and Fukushima did not yet use supervised backpropagation (BP) to train the weights of their nets by gradient descent in an objective function, such as the total classification error on a given training set of input patterns and corresponding labels, although BP was also developed back then.

BP's continuous form was derived in the early 1960s (Kelley 1960; Bryson 1961; Bryson and Ho 1969). Dreyfus (1962) published the elegant derivation of BP based on the chain rule only. BP's modern efficient version for discrete sparse networks (including FORTRAN code) was published by Linnainmaa (1970). Here the complexity of computing the derivatives of the output error with respect to each weight is proportional to the number of weights. That's the method still used today. Dreyfus (1973) used BP to change weights of controllers in proportion to such gradients. By 1980, automatic differentiation could derive BP for any differentiable graph (Speelpenning 1980). Werbos (1982) published the first application of BP to NNs, extending thoughts in his 1974 thesis, which did not yet have Linnainmaa's modern, efficient form of BP. In 1980–1990, computers became 10,000 times faster than those of 1960–1970 and widely accessible in academic labs. Computational experiments then demonstrated that BP in NNs can indeed yield useful internal representations in hidden layers of NNs (Rumelhart et al. 1986). Wan (1994) produced the first BP-trained NN to win a controlled pattern recognition contest with secret test set. Amari (1998) described BP for natural gradient-based NNs. By 2003, deep BP-based standard FNNs with up to seven layers were used to successfully classify high-dimensional data (e.g., Vieira and Barradas 2003).

In the 2000s, computing hardware had again become 10,000 times faster than in the 1980s. Cheap massively parallel graphics processing units (GPUs, originally developed for video games) started to revolutionize NN research. Standard FNNs implemented on GPU were 20 times faster than on CPU (Oh and Jung 2004). A plain GPU-based FNN trained by BP with pattern distortions (Baird 1990) set a new record of 0.35 % error rate (Ciresan et al. 2010) on the MNIST handwritten digit dataset, which by then had been the perhaps most famous machine learning benchmark for decades. This seemed to suggest that advances in exploiting modern computing hardware were more important than advances in algorithms.

Backpropagation for CNNs

LeCun et al. (1989) first applied BP to Neocognitron-like CNNs, achieving good performance on MNIST. Similar CNNs were used commercially in the 1990s. Ranzato et al. (2007) first applied BP to max-pooling CNNs (MPCNNs); advantages of doing this were pointed out subsequently (Scherer et al. 2010).

Efficient parallelized GPU-based MPCNNs (Ciresan et al. 2011) further improved the MNIST record dramatically, achieving human performance (around 0.2 %) for the first time (Ciresan et al. 2012c). To detect human actions in surveillance videos, a three-dimensional CNN, combined with support vector machines, was part of a larger system using a bag of features approach to extract regions of interest. The system won three 2009 TRECVID competitions. These were possibly the first official international contests won with the help of (MP)CNNs; compare (Ji et al. 2013).

In 2011, an ensemble (Breiman 1996; Schapire 1990) of GPU-based MPCNNs also was the first system to achieve superhuman visual pattern recognition in a controlled competition, namely, the IJCNN 2011 traffic sign recognition contest in Silicon Valley (Ciresan et al. 2012c). The system was twice better than humans and three times better than the nearest nonhuman competitor. Subsequently, similar committees of GPU-MPCNNs became widely used and also won the 2012 ImageNet classification contest (Krizhevsky et al. 2012), which is popular in the computer vision community. Further progress on ImageNet was achieved through variants of such systems (e.g., Zeiler and Fergus 2013; Szegedy et al. 2014; Simonyan and Zisserman 2015).

In 2012, a GPU-MPCNN committee also was the first deep learning NN to win a contest on visual object discovery in large images (Ciresan et al. 2013), namely, the ICPR 2012 Contest on Mitosis Detection in Breast Cancer Histological Images. Here deep MPCNNs are trained on labelled patches of big images and then used as feature detectors to be shifted across unknown visual scenes, using various rotations and zoom factors. Image parts that yield highly active output units are likely to contain objects similar to those the NN was trained on. A similar GPU-MPCNN committee was the first deep learner to win a pure image segmentation contest (Ciresan et al. 2012a), namely, the ISBI 2012 segmentation of neuronal structures in EM stacks challenge. The MPCNN learned to predict for each pixel whether it belongs to the background. Fast MPCNN image scanners avoid redundant computations and speed up naive implementations by up to three orders of magnitude (Masci et al. 2013), extending earlier efficient methods for CNNs without MP (Vaillant et al. 1994).

It is fair to say that deep GPU-CNNs have revolutionized computer vision. For example, GPU-MPCNNs helped to recognize multi-digit numbers in Google Street View images (Goodfellow et al. 2014b), where part of the NN was trained to count visible digits. Other successful recent CNN applications include scene parsing (Farabet et al. 2013), shadow detection (Khan et al. 2014), and video classification (Karpathy et al. 2014), to name a few.

Fundamental Deep Learning Problem and Unsupervised Pre-training of RNNs and FNNs

There are extensions of backpropagation (BP) for supervised RNNs (e.g., Williams 1989; Robinson and Fallside 1987; Werbos 1988). During training by "BP through time" (BPTT), the RNN is "unfolded" into an FNN that has essentially as many layers as there are time steps in the observed sequence of input vectors.

The drawbacks of BP and BPTT became obvious in 1991, when the vanishing/exploding gradient problem or "Fundamental Deep Learning Problem" was identified and analyzed (Hochreiter 1991): With standard activation functions, cumulative backpropagated error signals either shrink exponentially in the number of layers (or time steps) or grow out of bounds. The problem is most apparent in RNNs, the deepest of all NNs.

To some extent, Hessian-free optimization can alleviate the problem for FNNs (Moller 1993; Pearlmutter 1994) and RNNs (Martens and Sutskever 2011).

To overcome the vanishing gradient problem, an early generative model was proposed, namely, an unsupervised stack of RNNs called the neural history compressor (Schmidhuber 1992b). A first RNN uses unsupervised learning to predict its next input. Each higher level RNN tries to learn a compressed representation of the info in the RNN below, trying to minimize the description length (or negative log probability) of the data. The top RNN may then find it easy to classify the data by supervised learning. One can also "distill" the knowledge of a higher RNN (the teacher) into a lower RNN (the student) by forcing the lower RNN to predict the hidden units of the higher one. In the early 1990s, such systems could solve previously unsolvable "very deep learning" tasks involving hundreds of subsequent computational stages.

A conceptually very similar but FNN-based system was the deep belief network (DBN, Hinton and Salakhutdinov 2006), a stack of restricted Boltzmann machines (RBMs, Smolensky 1986) with a single layer of feature-detecting units. They can be trained by the contrastive divergence algorithm (Hinton 2002). At least in theory under certain assumptions, adding more layers improves a bound on the data's negative log probability (Hinton et al. 2006), equivalent to the data's description length – just like with the RNN history compressor above. A GPU-DBN implementation (Raina et al. 2009) was orders of magnitudes faster than previous CPU-DBNs; see also Coates et al. (2013). DBNs achieved good results on phoneme recognition (Mohamed and Hinton 2010). Autoencoder stacks (Ballard 1987) became a popular alternative way of pre-training deep FNNs in unsupervised fashion, before fine-tuning them through BP (e.g., Bengio et al. 2007).

Generally speaking, unsupervised learning (UL) can help to encode input data in a form advantageous for further processing. For example, FNNs may profit from pre-training by competitive UL prior to BP-based fine-tuning

(Maclin and Shavlik 1995). Many UL methods generate distributed, sparse representations of input patterns. Ideally, given an ensemble of input patterns, redundancy reduction through a deep NN will create a factorial code (a code with statistically independent components) of the ensemble (Barlow et al. 1989). Such codes may be sparse and can be advantageous for (1) data compression, (2) speeding up subsequent BP, and (3) trivializing the task of subsequent naive yet optimal Bayes classifiers. Methods for deep UL FNNs include hierarchical self-organizing Kohonen maps (e.g., Koikkalainen and Oja 1990), hierarchical Gaussian potential function networks (Lee and Kil 1991), layer-wise UL of feature hierarchies fed into SL classifiers (Behnke 1999), the self-organizing tree algorithm (Herrero et al. 2001), and nonlinear autoencoders (AEs) with five or more layers (e.g., Kramer 1991). Predictability minimization (Schmidhuber 1992c) searches for factorial codes through nonlinear feature detectors that fight nonlinear predictors, trying to become both as informative and as unpredictable as possible. Hierarchical CNNs in a Neural Abstraction Pyramid (e.g., Behnke 2003b) can be trained to reconstruct images corrupted by structured noise, thus enforcing increasingly abstract image representations in deeper and deeper layers.

In many applications of the 2000s, however, DBNs and other unsupervised methods were largely replaced by purely supervised FNNs, especially MPCNNs (see above). Here history repeated itself, because already in the 1990s, unsupervised RNN-based history compressors (see above) were largely replaced by purely supervised LSTM RNNs (see below).

Very Deep Learning in Supervised Sequence-Processing RNNs

Supervised long short-term memory (LSTM) RNNs have been developed since the 1990s (e.g., Hochreiter and Schmidhuber 1997b; Gers and Schmidhuber 2001; Graves et al. 2009). Parts of LSTM RNNs are designed such that backpropagated errors can neither vanish nor explode but flow backward in “civilized” fashion for thousands or even more steps. Thus, LSTM variants could learn previously unlearnable very deep learning tasks (including some unlearnable by the 1992 history compressor above) that require to discover the importance of (and memorize) events that happened thousands of discrete time steps ago, while previous standard RNNs already failed in case of minimal time lags of ten steps. It is possible to evolve good problem-specific LSTM-like topologies (Bayer et al. 2009).

Recursive NNs (Goller and Küchler 1996) generalize RNNs, by operating on hierarchical structures, recursively combining child representations into parent representations. Bidirectional RNNs (BRNNs) (Schuster and Paliwal 1997) are designed for input sequences whose starts and ends are known in advance, such as spoken sentences to be labeled by their phonemes. DAG-RNNs (Baldi and Pollastri 2003) generalize BRNNs to multiple dimensions. Recursive NNs, BRNNs, and DAG-RNNs unfold their full potential when combined with LSTM (Graves et al. 2009).

Particularly successful in competitions were stacks of LSTM RNNs (Fernandez et al. 2007b) trained by connectionist temporal classification (CTC, Graves et al. 2006), a gradient-based method for finding RNN weights that maximize the probability of teacher-given label sequences, given (typically much longer and more high-dimensional) streams of real-valued input vectors. CTC performs simultaneous segmentation (alignment) and recognition. In 2009, CTC-trained LSTM became the first RNN to win controlled international contests, namely, three competitions in connected handwriting recognition. Hannun et al. (2014) used CTC-trained RNNs to break a famous speech recognition benchmark record, without using any traditional speech processing methods such as hidden Markov models (HMMs) or Gaussian mixture models.

Unlike HMMs and previous RNNs, LSTM can learn to recognize context-sensitive languages. By 2007, LSTM had started to revolutionize speech recognition, outperforming traditional

HMMs in keyword spotting tasks (Fernandez et al. 2007b). By 2013, LSTM achieved best known results on the famous TIMIT phoneme recognition benchmark (Graves et al. 2013). Hybrids of traditional methods and LSTM RNNs obtained best known performance on large-vocabulary speech recognition (Sak et al.; Google 2014a; Li and Wu 2015). LSTM also helped to improve the state of the art in numerous other fields, including image caption generation (in conjunction with CNNs) (Vinyals et al.; Google 2014a), machine translation (Sutskever et al.; Google 2014), text-to-speech synthesis (Fan et al. 2015; Zen and Sak 2015, now available for Google Android), photo-real talking heads (Fan et al.; Microsoft 2015), syntactic parsing for natural language processing (Vinyals et al.; Google, 2014b), and many other applications. In 2015, CTC-trained LSTM dramatically improved Google Voice (by 49 %) and is now available to a billion smartphone users (Sak et al. 2015).

Gradient-based LSTM is no panacea though. Other methods sometimes outperformed LSTM at least on certain tasks (e.g., Jaeger 2004; Schmidhuber et al. 2007; Martens and Sutskever 2011; Zimmermann et al. 2012; Pascanu et al. 2013b; Koutnik et al. 2014). Several alternative RNN-related methods with fast memory control have been proposed over the decades (e.g., AMAmemory 2015).

Some Tricks to Improve NNs

BP-like methods can be used to search for "simple," low-complexity NNs with high generalization capability. For example, weight decay (e.g., Hanson and Pratt 1989) encourages near-zero weights, by penalizing large weights. Related weight priors are implicit in additional penalty terms (MacKay 1992) or in methods based on validation sets (e.g., Hastie and Tibshirani 1990). Similar priors (or biases towards simplicity) are implicit in constructive and pruning algorithms, e.g., layer-by-layer sequential network construction (e.g., Ivakhnenko 1971), input pruning (Moody 1992), unit pruning (e.g., Ivakhnenko 1971; Mozer and Smolensky 1989), weight pruning (e.g., LeCun et al. 1990b), fast and short weight matrix-computing programs (Schmidhuber 1997), and flat minimum search (FMS, Hochreiter and Schmidhuber 1999). DBN training can be improved (Cho et al. 2012) through Tikhonov-type regularization (Tikhonov et al. 1977). See also sparsity-enforcing methods mentioned earlier.

Dropout (Hinton et al. 2012b) removes units from NNs during training to improve generalization. It is closely related to older, biologically plausible techniques for adding noise to neurons or synapses during training (e.g., Hanson 1990). NNs with competing units (e.g., Schmidhuber 1989b; Maass 2000; Goodfellow et al. 2013) tend to outperform those with noncompeting units and avoid catastrophic forgetting through BP when training sets change over time (Srivastava et al. 2013).

The popular activation function f of rectified linear units (ReLUs) is $f(x) = x$ for $x > 0$; $f(x) = 0$ otherwise. ReLU NNs are useful for RBMs (Nair and Hinton 2010; Maas et al. 2013), outperformed sigmoidal activation functions in deep NNs (Glorot et al. 2011), and helped to obtain best results on several benchmark problems across multiple domains (e.g., Krizhevsky et al. 2012).

Many additional tricks for improving NNs have been described (e.g., Montavon et al. 2012; Schmidhuber 2015).

Consequences for Neuroscience

Artificial NNs (ANNs) can help to better understand biological NNs (BNNs). The feature detectors learned by single-layer visual ANNs are similar to those found in early visual processing stages of BNNs. Likewise, the feature detectors learned in deep layers of visual ANNs should be highly predictive of what neuroscientists will find in deep layers of BNNs. While the visual cortex of BNNs may use quite different learning algorithms, its objective function to be minimized may be rather similar to the one of visual ANNs. In fact, results obtained with relatively deep artificial NNs (e.g., Yamins et al. 2013) seem compatible with insights about the visual pathway in the primate cerebral cortex, which has been studied for many decades.

Deep Learning with Spiking Neurons?

Current deep NNs greatly profit from GPUs, which are little ovens, much hungrier for energy than biological brains, whose neurons efficiently communicate by brief spikes (e.g., Hodgkin and Huxley 1952) and often remain quiet. Many computational models of such spiking neurons have been proposed and analyzed (e.g., Gerstner and Kistler 2002). Future energy-efficient hardware for DL in NNs may implement aspects of such models – see numerous references in the survey (Schmidhuber 2015, Sect. 5.26). In practical applications, however, current artificial networks of spiking neurons cannot yet compete with the best traditional deep NNs.

Deep Reinforcement Learning (RL)

Reinforcement learning (RL) is the most general type of learning. General RL agents must discover, without the aid of a teacher, how to interact with a dynamic, initially unknown, partially observable environment in order to maximize their expected cumulative reward signals (e.g., Kaelbling et al. 1996; Sutton and Barto 1998; Wiering and van Otterlo 2012). There may be arbitrary, a priori unknown delays between actions and perceivable consequences. The RL problem is as hard as any problem of computer science, since any task with a computable description can be formulated in the general RL framework (e.g., Hutter 2005). Deep FNNs and RNNs are useful tools for various types of RL. Many references on this since the 1980s can be found in the recent survey (Schmidhuber 2015, Sect. 6).

Outlook

Deep learning in NNs is more than a temporary fad. Physics seems to dictate that any future efficient computational hardware will have to be brain-like, with many compactly placed processors in three-dimensional space, sparsely connected by many short and few long wires, to minimize total connection cost (even if the "wires" are actually light beams). The basic architecture is essentially the one of a deep, sparsely connected, three-dimensional RNN, and deep learning methods for such RNNs are expected to become even much more important than they are today.

The contents of this article may be used for educational and noncommercial purposes, including articles for Wikipedia and similar sites.

Recommended Reading

Aizenberg I, Aizenberg NN, Vandewalle JPL (2000) Multi-valued and universal binary neurons: theory, learning and applications. Springer, Boston. First work to introduce the term "Deep Learning" to Neural Networks

AMAmemory (2015) Answer at reddit AMA (Ask Me Anything) on "memory networks" etc (with references) http://www.reddit.com/r/MachineLearning/comments/2xcyrl/i_am_j%C3%BCrgen_schmidhuber_ama/cp0q12t

Amari S-I (1998) Natural gradient works efficiently in learning. Neural Comput 10(2):251–276

Baird H (1990) Document image defect models. In: Proceedings of IAPR workshop on syntactic and structural pattern recognition, Murray Hill

Baldi P, Pollastri G (2003) The principled design of large-scale recursive neural network architectures – DAG-RNNs and the protein structure prediction problem. J Mach Learn Res 4:575–602

Ballard DH (1987) Modular learning in neural networks. In: Proceedings of AAAI, Seattle, pp 279–284

Barlow HB, Kaushal TP, Mitchison GJ (1989) Finding minimum entropy codes. Neural Comput 1(3):412–423

Bayer J, Wierstra D, Togelius J, Schmidhuber J (2009) Evolving memory cell structures for sequence learning. In: Proceedings of ICANN, vol 2. Springer, Berlin/New York, pp 755–764

Behnke S (1999) Hebbian learning and competition in the neural abstraction pyramid. In: Proceedings of IJCNN, vol 2. Washington, pp 1356–1361

Behnke S (2003) Hierarchical neural networks for image interpretation. Lecture notes in computer science, vol LNCS 2766. Springer, Berlin/New York

Bengio Y, Lamblin P, Popovici D, Larochelle H (2007) Greedy layer-wise training of deep networks. In:

Cowan JD, Tesauro G, Alspector J (eds) Proceedings of NIPS 19, MIT Press, Cambridge, pp 153–160

Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140

Bryson AE (1961) A gradient method for optimizing multi-stage allocation processes. In: Proceedings of Harvard university symposium on digital computers and their applications, Harvard University Press, Cambridge

Bryson A, Ho Y (1969) Applied optimal control: optimization, estimation, and control. Blaisdell Publishing Company, Washington

Cho K, Ilin A, Raiko T (2012) Tikhonov-type regularization for restricted Boltzmann machines. In: Proceedings of ICANN 2012, Springer, Berlin/New York, pp 81–88

Ciresan DC, Meier U, Gambardella LM, Schmidhuber J (2010) Deep big simple neural nets for handwritten digit recogntion. Neural Comput 22(12):3207–3220

Ciresan DC, Meier U, Masci J, Gambardella LM, Schmidhuber J (2011) Flexible, high performance convolutional neural networks for image classification. In: Proceedings of IJCAI, pp 1237–1242

Ciresan DC, Giusti A, Gambardella LM, Schmidhuber J (2012a) Deep neural networks segment neuronal membranes in electron microscopy images. In: Proceedings of NIPS, Quebec City, pp 2852–2860

Ciresan DC, Meier U, Masci J, Schmidhuber J (2012b) Multi-column deep neural network for traffic sign classification. Neural Netw 32:333–338

Ciresan DC, Meier U, Schmidhuber J (2012c) Multi-column deep neural networks for image classification. In: Proceedings of CVPR 2012, Long preprint. arXiv:1202.2745v1 [cs.CV]

Ciresan DC, Giusti A, Gambardella LM, Schmidhuber J (2013) Mitosis detection in breast cancer histology images with deep neural networks. In: Proceedings of MICCAI, vol 2. Nagoya, pp 411–418

Coates A, Huval B, Wang T, Wu DJ, Ng AY, Catanzaro, B (2013) Deep learning with COTS HPC systems. In: Proceedings of ICML'13

Dechter R (1986) Learning while searching in constraint-satisfaction problems. University of California, Computer Science Department, Cognitive Systems Laboratory. First paper to introduce the term "Deep Learning" to Machine Learning; compare a popular G+ post on this. https://plus.google.com/100849856540000067209/posts/7N6z251w2Wd?pid=6127540521703625346&oid=100849856540000067209

Dreyfus SE (1962) The numerical solution of variational problems. J Math Anal Appl 5(1):30–45

Dreyfus SE (1973) The computational solution of optimal control problems with time lag. IEEE Trans Autom Control 18(4):383–385

Fan B, Wang L, Soong FK, Xie L (2015) Photo-real talking head with deep bidirectional LSTM. In: Proceedings of ICASSP 2015, Brisbane

Farabet C, Couprie C, Najman L, LeCun Y (2013) Learning hierarchical features for scene labeling. IEEE Trans Pattern Anal Mach Intell 35(8):1915–1929

Fernandez S, Graves A, Schmidhuber J (2007a) An application of recurrent neural networks to discriminative keyword spotting. In: Proceedings of ICANN, vol 2. pp 220–229

Fernandez S, Graves A, Schmidhuber J (2007b) Sequence labelling in structured domains with hierarchical recurrent neural networks. In: Proceedings of IJCAI

Fu KS (1977) Syntactic pattern recognition and applications. Springer, Berlin

Fukushima K (1979) Neural network model for a mechanism of pattern recognition unaffected by shift in position – neocognitron. Trans. IECE J62-A(10):658–665

Gers FA, Schmidhuber J (2001) LSTM recurrent networks learn simple context free and context sensitive languages. IEEE Trans Neural Netw 12(6):1333–1340

Gerstner W, Kistler WK (2002) Spiking neuron models. Cambridge University Press, Cambridge

Glorot X, Bordes A, Bengio Y (2011) Deep sparse rectifier networks. In: Proceedings of AISTATS, vol 15. Fort Lauderdale, pp 315–323

Goodfellow IJ, Warde-Farley D, Mirza M, Courville A, Bengio Y (2013) Maxout networks. In: Proceedings of ICML, Atlanta

Goodfellow IJ, Bulatov Y, Ibarz J, Arnoud S, Shet V (2014b) Multi-digit number recognition from street view imagery using deep convolutional neural networks. arXiv preprint arXiv:1312.6082 v4

Goller C, Küchler A (1996) Learning task-dependent distributed representations by backpropagation through structure. In: IEEE international conference on neural networks 1996, vol 1, pp 347–352

Graves A, Fernandez S, Gomez FJ, Schmidhuber J(2006) Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural nets. In: Proceedings of ICML'06, Pittsburgh, pp 369–376

Graves A, Liwicki M, Fernandez S, Bertolami R, Bunke H, Schmidhuber J (2009) A novel connectionist system for improved unconstrained handwriting recognition. IEEE Trans Pattern Anal Mach Intell 31(5):855–868

Graves A, Mohamed A-R, Hinton GE (2013) Speech recognition with deep recurrent neural networks. In: Proceedings of ICASSP, Vancouver, pp 6645–6649

Hannun A, Case C, Casper J, Catanzaro B, Diamos G, Elsen E, Prenger R, Satheesh S, Sengupta S, Coates A, Ng AY (2014) Deep speech: scaling up end-to-end speech recognition. arXiv preprint http://arxiv.org/abs/1412.5567

Hanson SJ, Pratt LY (1989) Comparing biases for minimal network construction with back-propagation. In: Touretzky DS (ed) Proceedings of NIPS, vol 1. Morgan Kaufmann, San Mateo, pp 177–185

Hanson SJ (1990) A stochastic version of the delta rule. Phys D: Nonlinear Phenom 42(1):265–272

Hastie TJ, Tibshirani RJ (1990) Generalized additive models, vol 43. CRC Press

Hebb DO (1949) The organization of behavior. Wiley, New York

Herrero J, Valencia A, Dopazo J (2001) A hierarchical unsupervised growing neural network for clustering gene expression patterns. Bioinformatics 17(2):126–136

Hinton G, Salakhutdinov R (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507

Hinton GE (2002) Training products of experts by minimizing contrastive divergence. Neural Comput 14(8):1771–1800

Hinton GE, Osindero S, Teh Y-W (2006) A fast learning algorithm for deep belief nets. Neural Comput 18(7):1527–1554

Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov RR (2012b) Improving neural networks by preventing co-adaptation of feature detectors. Technical report. arXiv:1207.0580

Hochreiter S (1991) Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut fuer Informatik, Lehrstuhl Prof. Brauer, Tech. Univ. Munich. Advisor: J. Schmidhuber

Hochreiter S, Schmidhuber J (1997a) Flat minima. Neural Comput 9(1):1–42

Hochreiter S, Schmidhuber J (1997b) Long short-term memory. Neural Comput 9(8):1735–1780. Based on TR FKI-207-95, TUM (1995)

Hochreiter S, Schmidhuber J (1999) Feature extraction through LOCOCODE. Neural Comput 11(3):679–714

Hodgkin AL, Huxley AF (1952) A quantitative description of membrane current and its application to conduction and excitation in nerve. J Physiol 117(4):500

Hutter M (2005) Universal artificial intelligence: sequential decisions based on algorithmic probability. Springer, Berlin

Ivakhnenko AG, Lapa VG (1965) Cybernetic Predicting Devices. CCM Information Corporation, New York

Ivakhnenko AG (1971) Polynomial theory of complex systems. IEEE Trans Syst Man Cybern (4):364–378

Jaeger H (2004) Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. Science 304:78–80

Ji S, Xu W, Yang M, Yu K (2013) 3D convolutional neural networks for human action recognition. IEEE Trans Pattern Anal Mach Intell 35(1): 221–231

Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: a survey. J AI Res 4:237–285

Karpathy A, Toderici G, Shetty S, Leung T, Sukthankar R, Fei-Fei L (2014) Large-scale video classification with convolutional neural networks. In: Proceedings of CVPR, Columbus

Kelley HJ (1960) Gradient theory of optimal flight paths. ARS J 30(10):947–954

Khan SH, Bennamoun M, Sohel F, Togneri R (2014) Automatic feature learning for robust shadow detection. In: Proceedings of CVPR, Columbus

Koikkalainen P and Oja E (1990) Self-organizing hierarchical feature maps. In: Proceedings of IJCNN, pp 279–284

Koutnik J, Greff K, Gomez F, Schmidhuber J (2014) A Clockwork RNN. In: Proceedings of ICML, vol 32. pp 1845–1853. arXiv:1402.3511 [cs.NE]

Kramer M (1991) Nonlinear principal component analysis using autoassociative neural networks. AIChE J 37:233–243

Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Proceedings of NIPS, Nevada, p 4

LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD (1989) Backpropagation applied to handwritten zip code recognition. Neural Comput 1(4):541–551

LeCun Y, Denker JS, Solla SA (1990b) Optimal brain damage. In: Touretzky DS (ed) Proceedings of NIPS 2, Morgan Kaufmann, San Mateo, pp 598–605

LeCun Y, Bengio Y, Hinton G (2015) Deep Learning. Nature 521:436–444. Link. See critique by J. Schmidhuber (2015) http://people.idsia.ch/~juergen/deep-learning-conspiracy.html

Lee S, Kil RM (1991) A Gaussian potential function network with hierarchically selforganizing learning. Neural Netw 4(2):207–224

Li X, Wu X (2015) Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition. In: Proceedings of ICASSP 2015. http://arxiv.org/abs/1410.4281

Linnainmaa S (1970) The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's thesis, University of Helsinki

Linnainmaa S (1976) Taylor expansion of the accumulated rounding error. BIT Numer Math 16(2):146–160

Maas AL, Hannun AY, Ng AY (2013) Rectifier nonlinearities improve neural network acoustic models. In: Proceedings of ICML, Atlanta

Maass W (2000) On the computational power of winner-take-all. Neural Comput 12:2519–2535

MacKay, DJC (1992) A practical Bayesian framework for backprop networks. Neural Comput 4:448–472

Maclin R, Shavlik JW (1995) Combining the predictions of multiple classifiers: using competitive learning to initialize neural networks. In: Proceedings of IJCAI, pp 524–531

Martens J, Sutskever I (2011) Learning recurrent neural networks with Hessian-free optimization. In: Proceedings of ICML, pp 1033–1040

Masci J, Giusti A, Ciresan DC, Fricout G, Schmidhuber J (2013) A fast learning algorithm for image segmentation with max-pooling convolutional networks. In: Proceedings of ICIP13, pp 2713–2717

McCulloch W, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys 7:115–133

Mohamed A, Hinton GE (2010) Phone recognition using restricted Boltzmann machines. In: Proceedings of ICASSP, Dallas, pp 4354–4357

Moller MF (1993) Exact calculation of the product of the Hessian matrix of feed-forward network error functions and a vector in O(N) time. Technical report PB-432, Computer Science Department, Aarhus University

Montavon G, Orr G, Mueller K (2012) Neural networks: tricks of the trade. Lecture notes in computer science, vol LNCS 7700. Springer, Berlin/Heidelberg

Moody JE (1992) The effective number of parameters: an analysis of generalization and regularization in nonlinear learning systems. In: Proceedings of NIPS'4, Morgan Kaufmann, San Mateo, pp 847–854

Mozer MC, Smolensky P (1989) Skeletonization: a technique for trimming the fat from a network via relevance assessment. In: Proceedings of NIPS 1, Morgan Kaufmann, San Mateo, pp 107–115

Nair V, Hinton GE (2010) Rectified linear units improve restricted Boltzmann machines. In: Proceedings of ICML, Dallas

Oh K-S, Jung K (2004) GPU implementation of neural networks. Pattern Recognit 37(6):1311–1314

Pascanu R, Mikolov T, Bengio Y (2013b) On the difficulty of training recurrent neural networks. In: ICML'13: JMLR: W&CP, vol 28

Pearlmutter BA (1994) Fast exact multiplication by the Hessian. Neural Comput 6(1):147–160

Raina R, Madhavan A, Ng A (2009) Large-scale deep unsupervised learning using graphics processors. In: Proceedings of ICML, Montreal, pp 873–880

Ranzato MA, Huang F, Boureau Y, LeCun Y (2007) Unsupervised learning of invariant feature hierarchies with applications to object recognition. In: Proceedings of CVPR, Minneapolis, pp 1–8

Robinson AJ, Fallside F (1987) The utility driven dynamic error propagation network. Technical report CUED/F-INFENG/TR.1, Cambridge University Engineering Department

Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65(6):386

Rumelhart DE, Hinton GE, Williams RJ (1986) Learning internal representations by error propagation. In: Rumelhart DE, McClelland JL (eds) Parallel distributed processing, vol 1, MIT Press, Cambridge, pp 318–362

Sak H, Senior AW, Beaufays F (2014) Long short-term memory recurrent neural network architectures for large scale acoustic modeling. INTERSPEECH

Sak H, Senior A, Rao K, Beaufays F, Schalkwyk J (2015) Google research blog. http://googleresearch.blogspot.ch/2015/09/google-voice-search-faster-and-more.html

Schapire RE (1990) The strength of weak learnability. Mach Learn 5(2):197–227

Scherer D, Mueller A, Behnke S (2010) Evaluation of pooling operations in convolutional architectures for object recognition. In: Proceedings of ICANN, Thessaloniki, pp 92–101

Schmidhuber J (1989b) A local learning algorithm for dynamic feedforward and recurrent networks. Connect Sci 1(4):403–412

Schmidhuber J (1992b) Learning complex, extended sequences using the principle of history compression. Neural Comput 4(2):234–242. Based on TR FKI-148-91, TUM, 1991

Schmidhuber J (1992c) Learning factorial codes by predictability minimization. Neural Comput 4(6):863–879

Schmidhuber J (1997) Discovering neural nets with low Kolmogorov complexity and high generalization capability. Neural Netw 10(5): 857–873

Schmidhuber J, Wierstra D, Gagliolo M, Gomez FJ (2007) Training recurrent networks by Evolino. Neural Comput 19(3):757–779

Schmidhuber J (2015) Deep learning in neural networks: an overview. Neural Netw 61:85–117. arXiv preprint 1404.7828

Schmidhuber J (2015) Deep learning. Scholarpedia 10(11):32832

Schuster M, Paliwal KK (1997) Bidirectional recurrent neural networks. IEEE Trans Signal Process 45:2673–2681

Sima J (1994) Loading deep networks is hard. Neural Comput 6(5):842–850

Simonyan K, Zisserman A (2015) Very deep convolutional networks for large-scale image recognition. arXiv preprint http://arxiv.org/abs/1409.1556

Smolensky P (1986) Parallel distributed processing: explorations in the microstructure of cognition, chapter information processing in dynamical systems: foundations of Harmony theory, vol 1. MIT Press, Cambridge, pp 194–281

Speelpenning B (1980) Compiling fast partial derivatives of functions given by algorithms. Ph.D. thesis, Department of Computer Science, University of Illinois, Urbana-Champaign

Srivastava RK, Masci J, Kazerounian S, Gomez F, Schmidhuber J (2013) Compete to compute. In: Proceedings of NIPS, Nevada, pp 2310–2318

Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: Proceedings of NIPS'2014. arXiv preprint arXiv:1409.3215 [cs.CL]

Sutton R, Barto A (1998) Reinforcement learning: an introduction. MIT Press, Cambridge

Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2014) Going deeper with convolutions. arXiv preprint arXiv:1409.4842 [cs.CV]

Tikhonov AN, Arsenin VI, John F (1977) Solutions of ill-posed problems. Winston, New York

Vaillant R, Monrocq C, LeCun Y (1994) Original approach for the localisation of objects in images. IEE Proc Vision Image Signal Process 141(4):245–250
Vieira A, Barradas N (2003) A training algorithm for classification of high-dimensional data. Neurocomputing 50:461–472
Vinyals O, Toshev A, Bengio S, Erhan D (2014a) Show and tell: a neural image caption generator. arXiv Preprint http://arxiv.org/pdf/1411.4555v1.pdf
Vinyals O, Kaiser L, Koo T, Petrov S, Sutskever I, Hinton G (2014b) Grammar as a foreign language. Preprint http://arxiv.org/abs/1412.7449
Wan EA (1994) Time series prediction by using a connectionist network with internal delay lines. In: Weigend AS, Gershenfeld NA (eds) Time series prediction: forecasting the future and understanding the past. Addison-Wesley, Reading, pp 265–295
Weng JJ, Ahuja N, Huang TS (1993) Learning recognition and segmentation of 3-d objects from 2-d images. Proceedings of the fourth international conference on computer vision. IEEE
Williams RJ (1989) Complexity of exact gradient computation algorithms for recurrent neural networks. Technical Report NU-CCS-89-27, Northeastern University, College of Computer Science, Boston
Wiering M, van Otterlo M (2012) Reinforcement learning. Springer, Berlin/Heidelberg
Werbos PJ (1974) Beyond regression: new tools for prediction and analysis in the behavioral sciences. Ph.D. thesis, Harvard University
Werbos PJ (1982) Applications of advances in nonlinear sensitivity analysis. In: Proceedings of the 10th IFIP conference, 31.8–4.9, NYC, pp 762–770
Werbos PJ (1988) Generalization of backpropagation with application to a recurrent gas market model. Neural Netw 1(4):339–356
Yamins D, Hong H, Cadieu C, DiCarlo JJ (2013) Hierarchical modular optimization of convolutional networks achieves representations similar to macaque IT and human ventral stream. In: Proceedings of NIPS, Nevada, pp 1–9
Zeiler MD, Fergus R (2013) Visualizing and understanding convolutional networks. Technical report arXiv:1311.2901 [cs.CV], NYU
Zen H, Sak H (2015) Unidirectional long short-term memory recurrent neural network with recurrent output layer for low-latency speech synthesis. In: Proceedings of ICASSP, Brisbane, pp 4470–4474
Zimmermann H-G, Tietz C, Grothmann R (2012) Forecasting with recurrent neural networks: 12 tricks. In: Montavon G, Orr GB, Mueller K-R (eds) Neural networks: tricks of the trade, 2nd edn. Lecture Notes in Computer Science, vol 7700. Springer, Berlin/New York, pp 687–707

Density Estimation

Claude Sammut
The University of New South Wales, Sydney,
NSW, Australia

Synonyms

Kernel density estimation

Definition

Given a set of observations, $x_1, \ldots, x_N$, which is a random sample from a probability density function $f_X(x)$, density estimation attempts to approximate $f_X(x)$ by $\widehat{f}_X(x_0)$.

A simple way of estimating a probability density function is to plot a histogram from a random sample drawn from the population. Usually, the range of data values is subdivided into equally sized intervals or *bins*. How well the histogram estimates the function depends on the bin width and the placement of the boundaries of the bins. The latter can be somewhat improved by modifying the histogram so that fixed boundaries are not used for the estimate. That is, the estimate of the probability density function at a point uses that point as the centre of a neighborhood. Following Hastie et al. (2009), the estimate can be expressed as:

$$\widehat{f_X}(x_0) = \frac{\#x_i \in N(x_0)}{N\lambda} \tag{1}$$

where $x_1, \ldots, x_N$ is a random sample drawn from a probability density function $f_X(x)$ and $\widehat{f_X}(x_0)$ is the estimate of f_X at point x_0. $N(x_0)$ is a neighborhood of width λ, around x_0. That is, the estimate is the normalized count of the number of values that fall within the neighborhood of x_0.

The estimate above is still *bumpy*, like the histogram. A smoother approximation can be obtained by using a *kernel function*. Each x_i in the sample is associated with a kernel function, usually Gaussian. The count in formula (1) above is replaced by the sum of the kernel function applied to the points in the neighborhood of x_0:

$$\widehat{f_X}(x_0) = \frac{1}{N\lambda}\sum_{i=1}^{N} K_\lambda(x_0, x_i) \qquad (2)$$

where K is the kernel function associated with sample x_i near x_0. This is called the *Parzen* estimate (Parzen 1962). The *bandwidth*, λ, affects the roughness or smoothness of the kernel histogram. The kernel density estimate is said to be under-smoothed if the bandwidth is too small. The estimate is over-smoothed if the bandwidth is too large.

Density estimation is most often used in association with memory-based classification methods, which can be thought of as weighted ▸ nearest neighbor classifiers.

▸ Mixture models and ▸ Locally weighted regression are forms of kernel density estimation.

Cross-References

- ▸ Kernel Methods
- ▸ Locally Weighted Regression for Control
- ▸ Mean Shift
- ▸ Mixture Model
- ▸ Nearest Neighbor
- ▸ Support Vector Machines

Recommended Reading

Kernel Density estimation is well covered in texts including Hastie et al. (2009), Duda et al. (2001) and Ripley (1996)

Duda RO, Hart PE, Stork DG (2001) Pattern classification, 2nd edn. Wiley, New York

Hastie T, Tibshirani R, Friedman J (2009) The elements of statistical learning: data mining, inference and perception, 2nd edn. Springer, New York

Parzen E (1962) On the estimation of a probability density function and the mode. Ann Math Stat 33:1065–1076

Ripley BD (1996) Pattern recognition and neural networks. Cambridge University Press, Cambridge

Density-Based Clustering

Joerg Sander
University of Alberta, Edmonton, AB, Canada
Statistical Machine Learning Group, NICTA,
Canberra, ACT, Australia

Abstract

The chapter gives a concise explanation of the basic principles of density-based clustering and points out important ”milestone papers” in this area.

Synonyms

Estimation of density level sets; Mode analysis; Nonparametric cluster analysis

Definition

Density-based clustering refers to unsupervised learning methods that identify distinctive groups/clusters in the data, based on the idea that a cluster in a data space is a contiguous region of high point density, separated from other such clusters by contiguous regions of low point density. The data points in the separating regions of low point density are typically considered noise/outliers.

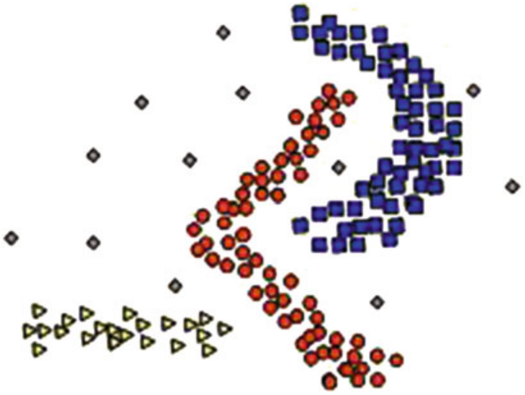
Motivation and Background

Clustering in general is an unsupervised learning task that aims at finding distinct groups in data, called “clusters.” The minimum requirements for this task are that the data is given as some set of objects O for which a dissimilarity-distance function $d: O \times O \rightarrow R^+$ is given. Often, O is a set of d-dimensional real-valued points, $O \subset R^d$, which can be viewed as a sample from some unknown probability density $p(x)$, with d as the Euclidean or some other form of distance.

There are different approaches to characterizing what establishes distinct groups in the data.

From a procedural point of view, many clustering methods try to find a partition of the data into k groups so that the within-cluster dissimilarities are minimized, while the between-cluster dissimilarities are maximized. The notions of within-cluster and between-cluster dissimilarity are defined using the given distance function d. Such methods correspond, from a statistical point of view, to a parametric approach where the unknown density $p(x)$ of the data is assumed to be a mixture of k densities $p_i(x)$, each corresponding to one of the k groups in the data; the $p_i(x)$ are assumed to come from some parametric family (e.g., Gaussian distributions) with unknown parameters, which are then estimated from the data.

In contrast, *density-based* clustering is a nonparametric approach where the groups in the data are considered to be the high-density areas of the density $p(x)$. Density-based clustering methods do not require the number of clusters as input parameters, nor do they make assumptions about the underlying density $p(x)$ or the variance within the groups that may exist in the data. Consequently, density-based clusters are not necessarily groups of points with high within-cluster similarity as measured by the distance function d but can have "arbitrary shape" in the feature space; they are sometimes also referred to as "natural clusters." This property makes density-based clustering particularly suitable for applications where clusters cannot be well described as distinct groups of low within-cluster dissimilarity, as, for instance, in spatial data where clusters of points in the space may form along natural structures such a rivers, roads, seismic faults, etc. Figure 1 illustrates density-based clusters using two-dimensional example, where the assumed dissimilarity function between the points is the Euclidean distance: there are three clusters indicated by triangles, points, and rectangles, as well as some noise points, indicated by diamond shapes. Note that the distance between some points within the clusters is much larger than the distance between some points from different clusters, yet the regions containing the clusters have clearly a higher point density than the region between them, and they can easily be separated.



Density-Based Clustering, Fig. 1 Illustration of a density-based clustering, showing three distinguishable groups

Density-based clustering is one of the prominent paradigms for clustering large data sets in the data mining community. It has been extensively studied and successfully used in many applications.

Structure of Learning System

Assuming that the data set $O \subset R^d$ is a sample from some unknown probability density $p(x)$, there are different ways of determining high-density areas of the density $p(x)$. Commonly, the notion of a high-density area is (implicitly or explicitly) based on a local density estimate at each point (typically some kernel or nearest neighbor density estimate) and a notion of connection between objects (typically points are connected if they are within a certain distance ε from each other); clusters are essentially constructed as maximal sets of objects which are directly or transitively connected to objects whose density exceeds some threshold λ. The set {x | p(x) > λ} of all high-density objects is called the *density level set* of p at λ. Objects that are not part of such clusters are called *noise* or *outliers*.

Different proposed density-based methods distinguish themselves mainly by how the density $p(x)$ is estimated, how the notion of connectivity is defined, and how the algorithm for finding connected components of the induced graph is implemented and supported by suitable data structures to achieve scalability for large data sets. Some methods include in a cluster only objects whose density exceed the threshold λ; others also include objects with lower density if they are connected to an object with density above the threshold λ.

Density-based clustering was probably introduced the first time by Wishart (1969). His algorithm for *one level mode analysis* consists of six steps: “(a) Select a distance threshold r, and a frequency (or density) threshold k. (b) Compute the triangular similarity matrix of all inter-point distances. (c) Evaluate the frequency k_i of each data point, defined as the number of points which lie within a distance r of point i (…). (d) Remove the ‘noise’ or non-dense points, those for which $k_i < k$. (e) Cluster the remaining dense points ($k_i > k$) by single linkage, forming the mode nuclei. (f) Reallocate each non-dense point to a suitable cluster according to some criterion (…).” (Wishart 1969).

Hartigan (1975) suggested a more general definition of a density-based cluster, a *density contour cluster* at level λ, as a maximally connected set of points x for which $p(x) > \lambda$, given a density $p(x)$ at each point x, a density threshold λ, and links specified for some pairs of objects. For instance, given a particular distance function, points can be defined as linked if the distance between them is no greater than some threshold r, or, if only direct links are available, one can define a “distance” for pairs of objects x and y in the following way:

$$d(x,y)=\begin{cases} -\min[p(x),p(y)] & x \text{ and } y \text{ are linked} \\ 0 & \text{otherwise} \end{cases}$$

To compute the density contour clusters, Hartigan, like Wishart, suggest a version of single-linkage clustering, which will construct the maximal connected sets of objects of density greater than the given threshold λ.

The DBSCAN algorithm (Ester et al. 1996) introduced density-based clustering independently to the Computing Science Community, also proposing the use of spatial index structures to achieve a scalable clustering algorithm. Assuming a distance threshold r, and a density threshold k, DBSCAN, like Wishart’s method, estimates the density for each point x_i as the number k_i of points that lie inside a radius r around x. *Core points* are defined as data points for which $k_i > k$. Points are considered directly connected if the distance between them is no greater than r. Density-based clusters are defined as maximally connected components of the set of points that lie within distance r from some core object (i.e., a cluster may contain points x_i with $k_i < k$, called *border objects*, if they are within distance r of a core object of that cluster). Objects not part of a cluster are considered as *noise*. The algorithm DBSCAN constructs clusters iteratively, starting a new cluster C with a non-assigned core object x and assigning all points to C that are directly or transitively connected to x. To determine directly and transitively connected points for a given point, a spatial index structure is used to perform range queries with radius r for each object that is newly added to a current cluster, resulting in an algorithm that performs well in practical situations when spatial index structures are effective (typically for low- to medium dimensional data), and has quadratic worst case runtime when index structures are not effective (e.g., for high-dimensional data).

DENCLUE (Hinneburg and Keim 1998) proposed a notion of density-based clusters using kernel density estimation. Each data point x is associated with (“attracted by”) a local maximum (“density attractor”) of the overall density function that lies in the direction of maximum increase in density from x. Density-based clusters are defined as connected components of density attractors with their associated points whose density estimate is above a given threshold λ. In this formulation, DBSCAN and Wishart's method can be seen as special cases of DENCLUE, using a uniform spherical kernel and, for Wishart's method, not including attracted points whose density is below λ. DENCLUE essentially uses a truncated Gaussian kernel for the implementation, which is based on a clever data structure to speed up local density estimation. The data space is partitioned into d-dimensional cells; nonempty cells are mapped to one-dimensional keys which are stored together with some sufficient statistics about the cell (number of points, pointers to points, and linear sum of the points belonging to the cell) in a search tree for efficient retrieval of neighboring cells and local density estimation (Hinneburg and Keim (1998) report that in an experimental comparison on 11-dimensional data sets of different sizes, DENCLUE runs up to 45 times faster than DBSCAN).

A large number of related methods and extensions have been proposed, particularly in computing science and application-oriented domains, some motivated by algorithmic considerations that could improve efficiency of the computation of density-based clusters, others motivated by special applications, proposing essentially density-based clustering algorithms using specific density measures and notions of connectivity. An algorithmic framework, called GDBSCAN, that generalizes the topological properties of density-based clusters can be found in Sander et al. (1998). GDBSCAN generalizes the notion of a density-based clustering to that of a *density-connected decomposition*, assuming only a reflexive and symmetric *neighborhood* relation for pairs of objects (direct links between some objects), and an arbitrary predicate, called "*MinWeight*," that evaluates to *true* for some neighborhood sets of objects and *false* on others, a core object can be defined as an object whose neighborhood satisfies the *MinWeight* predicate. Then, a density-connected decomposition consists of the maximally connected components of the set of objects that are in the neighborhood of some core object, and they can be computed with the same algorithmic scheme as density-based clusters by DBSCAN.

One of the principal problems of finding the density-based clusters of a density level set for a single level λ is how to determine a suitable level λ. The result of a density-based clustering method depends critically on the choice of λ, which may be difficult to determine even in situations when a meaningful level exists, depending on how well the clusters are separated in the given sample. In other situations, it may not even be possible to characterize the cluster structure appropriately using a single density threshold, when modes exist in different regions of the data space that have very different local densities or when clusters are nested within clusters. The problem of selecting suitable density threshold parameters has been already observed by Wishart (1969) who also proposed a hierarchical algorithm to represent the clusters at different density levels. Hartigan (1975) also observed that density-based clusters at different density levels have a hierarchical structure, a *density contour tree*, based on the fact that two clusters (i.e., connected components) of different density levels are either disjoint or the cluster of higher density is completely contained in the cluster of lower density. Recent proposals for hierarchical clustering methods based on a density estimate and a notion of linkage are, e.g., Ankerst et al. (1999), Stuetzle (2003), and Campello et al. (2013). These hierarchical methods are closely related and are essentially processing and rendering a minimum spanning tree of the data –with edge weights defined in different ways– and are thus also closely related to single-linkage clustering. Hierarchical methods do not, in a strict sense, compute a partition of the data but compute a representation of the overall hierarchical density structure of the data from which particular density-based clusters at different density levels or a global density threshold (a "cut level") could be determined. Recent work (Campello et al. 2013) provides an efficient hierarchical version DBSCAN, called HDBSCAN, which includes a method for automatically extracting a flat partitioning from possibly different levels of a density-based clustering hierarchy, containing only significant clusters according to a cluster stability measure.

Cross-References

▶ Clustering
▶ Density Estimation

Recommended Reading

Ankerst M, Breunig MM, Kriegel H-P, Sander J (1999) OPTICS: ordering points to identify the clustering structure. In: Delis A, Faloutsos C, Ghandeharizadeh S (eds) Proceedings of the 1999 ACM SIGMOD international conference on management of data, Philadelphia

Campello RJGB, Moulavi D, Sander J (2013) Density-based clustering based on hierarchical density estimates. In: Proceedings of the 17th Pacific-Asia conference on knowledge discovery in databases, PAKDD 2013. Lecture notes in computer science, vol 7819, p 160

Ester M, Kriegel H-P, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Simoudis E, Han J, Fayyad UM (eds) Proceedings of the 2nd international conference on knowledge discovery and data mining, Portland

Hartigan JA (1975) Clustering algorithms. Wiley, New York

Hinneburg A, Keim DA (1998) An efficient approach to clustering in large multimedia databases with noise. In: Agrawal R, Stolorz P (eds) Proceedings of the 4th international conference on knowledge discovery and data mining, New York City

Sander J, Ester M, Kriegel H-P, Xu X (1998) Density-Based clustering in spatial databases: the algorithm GDBSCAN and its applications. Data Min Knowl Discov 2(2):169–194

Stuetzle W (2003) Estimating the cluster tree of a density by analyzing the minimal spanning tree of a sample. J Classif 20(1):025–047

Wishart D (1969) Mode analysis: a generalization of nearest neighbor which reduces chaining effects. In: Numerical Taxonomy, ed. A. J. Cole, London: Academic Press, 282–311

Dependency Directed Backtracking

▶ Intelligent Backtracking

Detail

In ▶ Minimum Message Length, *detail* is the code or language shared between sender and receiver that is used to describe the data conditional on the asserted model.

Diagonal Matrix

▶ *K*-Way Spectral Clustering

Differential Prediction

▶ Uplift Modeling

Digraphs

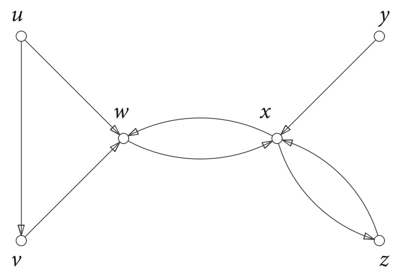
Synonyms

Directed graphs

Definition

A *digraph* D consists of a (finite) set of *vertices* $V(D)$ and a set $A(D)$ of ordered pairs, called *arcs*, of distinct vertices. An arc (u, v) has *tail* u and *head* v, and it is said to leave u and enter v.

Figure 1 shows a digraph D with vertex set $V(D) = \{u, v, w, x, y, z\}$ and arc set $A(D) = \{(u, v), (u, w), (v, w), (w, x), (x, w), (x, z), (y, x),$



Digraphs, Fig. 1 A digraph

$(z, x)\}$. Digraphs can be viewed as generalizations of ▶ graphs.

Dimensionality Reduction

Michail Vlachos
IBM Research, Zurich, Switzerland

Abstract

Dimensionality reduction in an important data pre-processing when dealing with Big Data. We explain how it can be used for speeding up search operation and show applications for time-series datasets.

Synonyms

Feature selection; Feature projection; lossy compression

Introduction

Every data object in a computer is represented and stored as a set of features, for example, color, price, dimensions, and so on. Instead of the term *features*, one can interchangeably use the term *dimensions* because an object with n features can also be represented as a multidimensional point in an n-dimensional space. Therefore, dimensionality reduction (dR) refers to the process of mapping an n-dimensional point into a lower k-dimensional space. This operation reduces the size for representing and storing an object or a dataset in general; hence, dimensionality reduction can be seen as a method for data compression. In addition, this process promotes data visualization, particularly when objects are mapped onto two or three dimensions. Finally, in the context of classification, dimensionality reduction can be a useful tool for (a) making tractable classification schemes that are superlinear with respect to dimensionality tractable, (b) reducing the variance of classifiers that are plagued by large variance in higher dimensionalities, and (c) removing the noise that may be present, thus boosting classification accuracy.

Motivation and Background

There are many techniques for dimensionality reduction. The objective of these techniques is to appropriately select the k dimensions (and also the number k) so that the important characteristics of the original object are retained. For example, when performing dimensionality reduction on an image, e.g., using a wavelet-based technique, the desirable outcome is that the difference between the original and the final images is almost imperceptible.

When performing dimensionality reduction not on a single object, but on a dataset, an additional requirement is that the relationship between the objects in the original space be preserved. This is particularly important for reasons of classification and visualization in the new space.

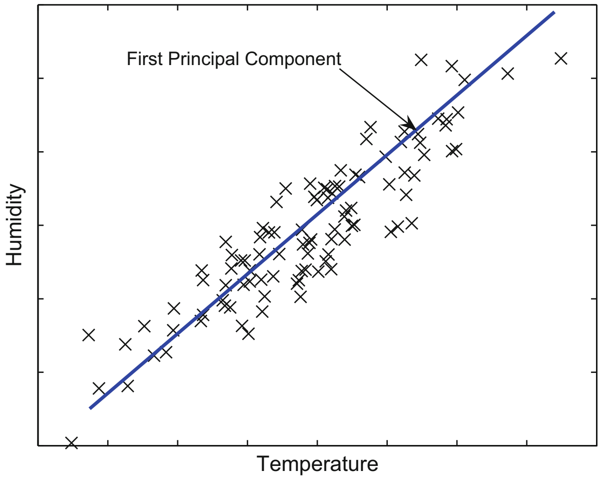
Two important categories of dimensionality reduction techniques exist:

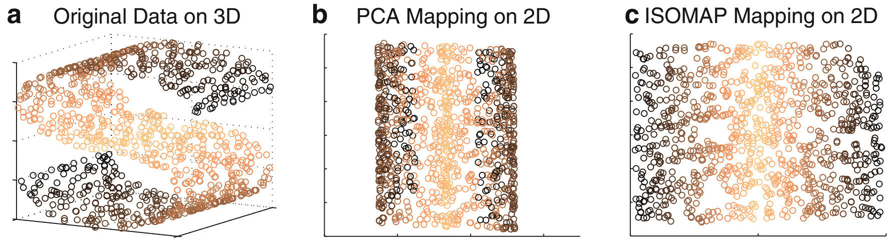
- **Feature selection** techniques, in which only the most important or descriptive features/dimensions are retained, and the rest are discarded. More details on such techniques can be found under the entry ▶ Feature Selection
- **Feature projection** methodologies, which project the existing features onto different dimensions or axes. The aim here is, again, to find those new data axes that retain the dataset structure and preserve its variance as closely as possible.

Feature projection techniques typically exploit the correlations between the various data dimensions, with the goal of creating dimensions/axes that are uncorrelated and sufficiently describe the data.

One of the most popular dimensionality reduction techniques is *principal component analysis* or PCA. It attempts to discover those axes (or

Dimensionality Reduction, Fig. 1 Principal component analysis





Dimensionality Reduction, Fig. 2 Nonlinear dimensionality reduction techniques produce a better low-dimensional data mapping than PCA if the original data lie on a high-dimensional manifold

components) onto which the data can be projected while maintaining the original correlation between the dimensions. Consider, for example, a dataset that contains records of environmental measurements over a period of time, such as humidity and temperature. The two attributes can be highly correlated, as shown in Fig. 1. By deploying PCA, this trend will be discovered, and the original two-dimensional points can be reduced to one-dimensional by projecting the original points onto the first *principal component*. In that way, the derived dataset can be stored in less space.

PCA uses the Euclidean distance as the measure of dissimilarity among objects. The first principal component (or axis) indicates the direction of maximum variance in the original dimensions. The second component shows the direction of the next highest variance (and is uncorrelated to the first component), etc.

Other dimensionality reduction techniques optimize or preserve other criteria than PCA does. Manifold-inspired methods such as ISOMAP (Tenenbaum et al. 2000) preserve the geodesic distances between objects. The notion here is to approximate the distance between objects “through” the remaining ones. The result of such dimensionality reduction techniques is that when the data lie on a manifold, the projected dimensions effectively “unfold” the underlying high-dimensional manifold. An example of this mapping is illustrated in Fig. 2, where it is also compared with the respective PCA mapping.

Other recent dimensionality reduction techniques include locally linear embedding (LLE) (Roweis and Saul 2000) and Laplacian eigenmaps (Belkin and Niyogi 2002). We also refer the interested practitioner to van der Maaten et al. (2009), for a detailed comparison of various techniques and also for Matlab implementations on a variety of dimensionality reduction algorithms.

In general, dimensionality reduction is a commonly practiced and useful operation in database and machine-learning systems because it offers the following desirable properties:

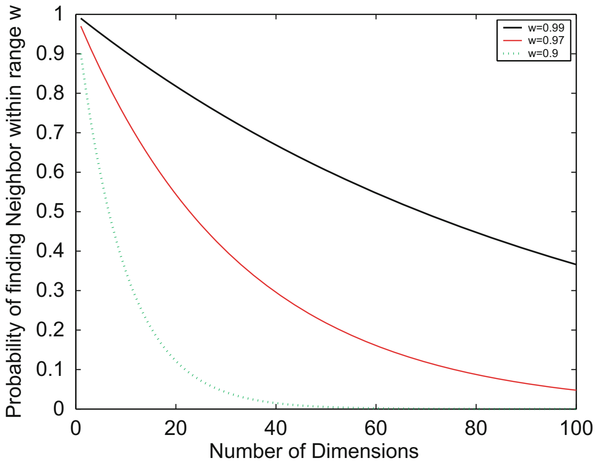
- Data compression: the dataset objects are represented in fewer dimensions, hence saving important disk storage space and offering faster loading of the compressed data from the disk.
- Better data visualization: the relationships between the original high-dimensional objects can be visualized in two- or three-dimensional projections.
- Improved classification accuracy: this can be attributed to both variance reduction and noise removal from the original high-dimensional dataset.
- More efficient data retrieval: dimensionality reduction techniques can also assist in making the retrieval of the original uncompressed data faster and more efficient, by offering very fast prefiltering with the help of the compressed data representation.
- High index performance: more effective use of indexing structures can be achieved by using the compressed data, because indexing techniques only work efficiently with lower-dimensional data (e.g., from 1 to 30 dimensions, depending on the type of the index).

The fact that indexing structures do not perform efficiently for higher-dimensional data is also known as the ▶ Curse of Dimensionality. Suppose that we are interested in performing search operations on a set of high-dimensional data. For simplicity, let us assume that the data lie in a unit hypercube $C = [0, 1]^d$, where d is the data dimensionality. Given a query point, the probability P_w that a match (neighbor) exists within radius w in the data space of dimensionality d is given by $P_w(d) = w^d$.

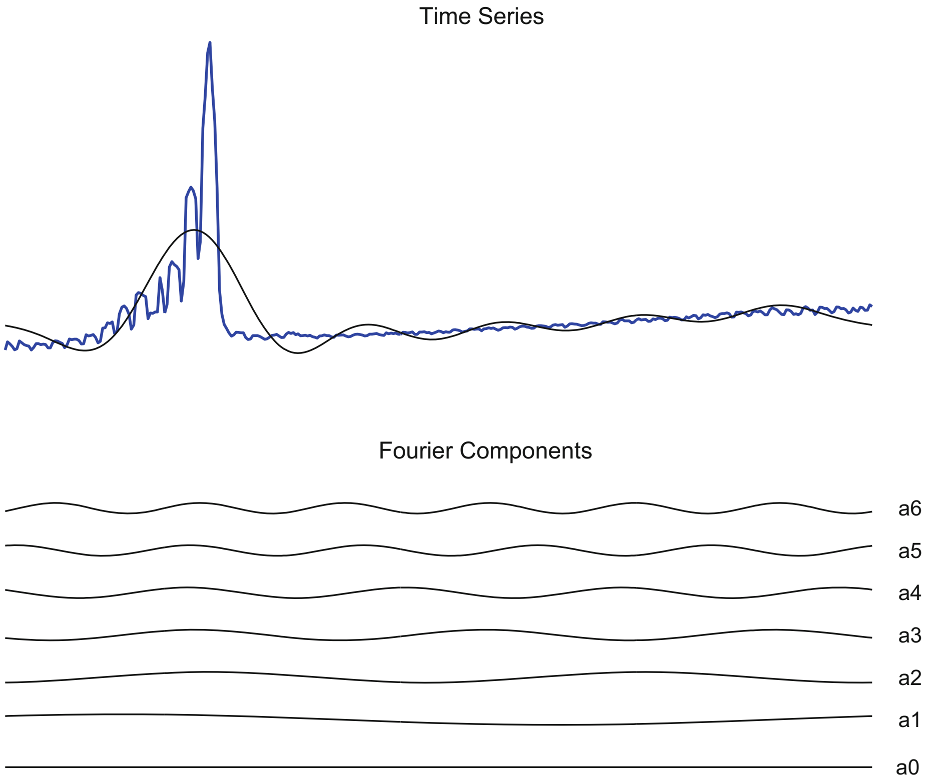
Figure 3 illustrates this probability for various values of w. Evidently, at higher dimensionalities the data becomes very sparse, and even at large radii, only a small portion of the entire space is covered. In simple terms the "curse of dimensionality" translates into the following fact: for large dimensionalities, existing indexing structures outperform a linear scan of all the data, only when the dataset size (number of objects) grows exponentially with respect to the dimensionality.

Applications: Dimensionality Reduction for Time-Series Data

In this section, we provide more detailed examples of dimensionality reduction techniques for time-series data. We chose time series to convey visually the effect of dimensionality reduction particularly for high-dimensional data such as



Dimensionality Reduction, Fig. 3 Probability $P_w(d)$ against dimensionality d. The data becomes sparse in higher dimensions



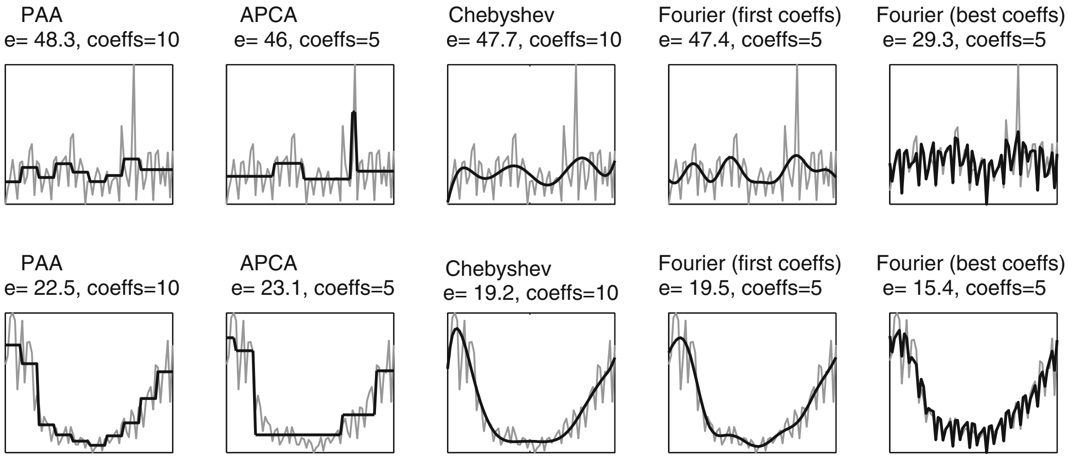
Dimensionality Reduction, Fig. 4 Decomposition of a signal into the first 7 Fourier coefficients. We can see that by using even only few of the Fourier coefficients we can achieve a good reconstruction of the original signal

time series. Later, we also show how dimensionality reduction on large datasets can help speed up search operations over the original uncompressed data.

Dimensionality reduction for one- and two-dimensional signals is commonly accomplished using Fourier decomposition. This method for data representation was first presented in the beginning of the nineteenth century by Jean Baptiste Fourier (1768–1830), in his seminal work "On the Propagation of Heat in Solid Bodies." Fourier came to the conclusion that every function could be expressed as a sum of trigonometrical series (i.e., sines and cosines). This original work was initially met with doubt (even by famous mathematicians such as Lagrange and Laplace), because of its unexpected result, and moreover, the solution was considered impractical because of the complex integration functions.

However, in the twentieth century, no one can deny the importance of Fourier's findings. With the introduction of fast ways to compute the Fourier decomposition in the 1960s (fast Fourier transform or FFT), the barrier of the high computational complexity was lifted. What the Fourier transform attempts to achieve is to represent the original signal as a linear combination of sinusoids. Therefore, each Fourier coefficient is a complex number that essentially encodes the amplitude and the phase of each of these sinusoids, after the original signal has been projected on them.

For most signals, the original sequence can be reconstructed with high accuracy using just few of the coefficients. This is where the great power of the Fourier transformation lies: by neglecting the majority of the coefficients, we can essentially compress the signal or describe it with fewer numbers. For stock market data or other



Dimensionality Reduction, Fig. 5 Comparison of various dimensionality reduction techniques for time-series data. The *darker line* indicates the approximation using the number of coefficients reported. Each figure also shows the error *e* introduced by the dimensionality reduction technique. Lower errors indicate better low-dimensional approximation of the original object

time series that follow the pattern of a random walk, the first few coefficients, which capture the low frequencies of the signal, are sufficient to describe the signal accurately (or, equivalently, to capture most of its energy). Figure 4 depicts a signal of 1024 points and its reconstruction using 7 Fourier coefficients (i.e., using $7 \times 2 = 14$ numbers).

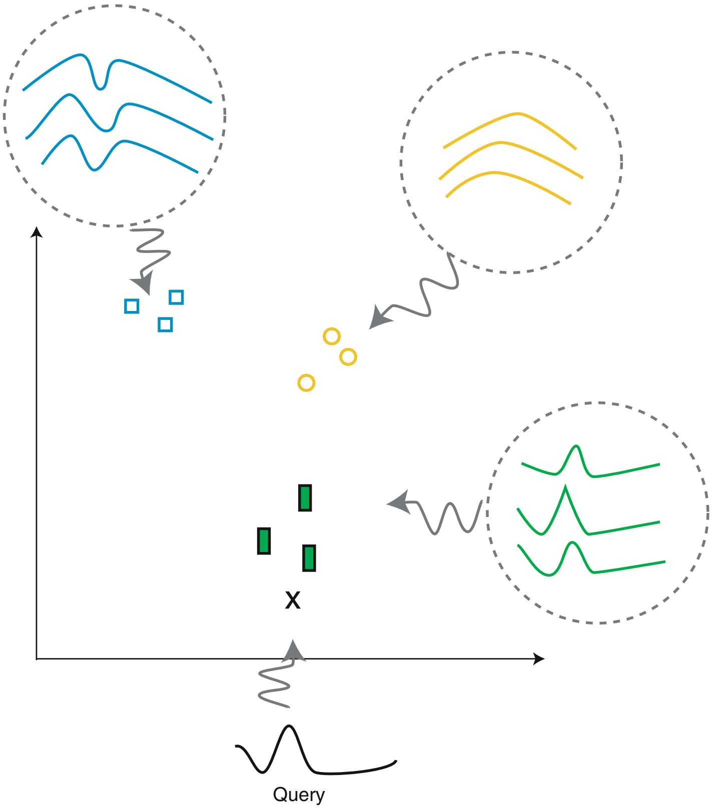
Other popular dimensionality reduction techniques for time-series data are the various wavelet transforms; piecewise linear approximations; piecewise aggregate approximation (PAA), which can be regarded as a projection in time of the wavelet coefficients adaptive piecewise constant approximation (APCA Keogh et al. 2001) and uses the highest energy wavelet coefficients; Chebyshev polynomial approximation symbolic approximation of time series (such as the SAX representation Lin et al. 2003).

No dimensionality reduction technique is universally better than all the others. Depending on the dataset characteristics, one method may provide a better approximation of a dataset than the other techniques. Therefore, the key is to carefully pick the representation that best suits the specific application or the task at hand. In Fig. 5, we demonstrate various dimensionality reduction techniques and the quality of the time-series approximation. For all methods, the same storage space is allocated for the compressed sequences. The time-series reconstruction is shown in a darker color, and the approximation error to the original sequence is also reported. In general, we notice that dimensionality reduction techniques based on selection of the highest energy coefficients consistently provide a high-quality sequence approximation.

Dimensionality Reduction and Lower Bounding

Dimensionality reduction can be a useful tool for speeding up search operations. Figure 6 illustrates dimensionality reduction for high-dimensional time-series data. After dimensionality reduction, each object is represented using fewer dimensions (attributes), so it is represented in a lower-dimensional space. Then, suppose that a user poses another high-dimensional object as query and wishes to find all the objects closest to this query.

To avoid the search on the original high-dimensional space, the query is also transformed into a point in the lower-dimensional space, and its closest matches can be discovered in the vicinity of the projected query point. However,



Dimensionality Reduction, Fig. 6 Search and dimensionality reduction. Every object (time series in this case) is transformed into a lower-dimensional point. User queries are also projected into the new space. Similarity search consists of finding the closest points to the query projection

when searching using the compressed objects, one needs to provide an estimate of the distance between the original objects. Typically, it is preferable that the distance in the new space underestimates (or lower bounds) the distance in the original high-dimensional space. The reason for this is the following.

Suppose that we are seeking the 1-Nearest-Neighbor (1-NN) a query Q in a database $\mathcal{D}$. By examining all objects (linear scan), one can guarantee that the best match will be found. Can one provide the same guarantee (i.e., that the same best match will be returned) when examining the compressed objects (after dimensionality reduction)?

The answer is positive, as long as the distance on the compressed data *underestimates* or *lower bounds* the distance on the raw data. In other words, the dimensionality reduction (dR) that is performed on the raw data must have the following property:

$$\text{Having}\quad A \subset \mathcal{D} \xrightarrow{dR} a \quad \text{and} \quad Q \xrightarrow{dR} q$$

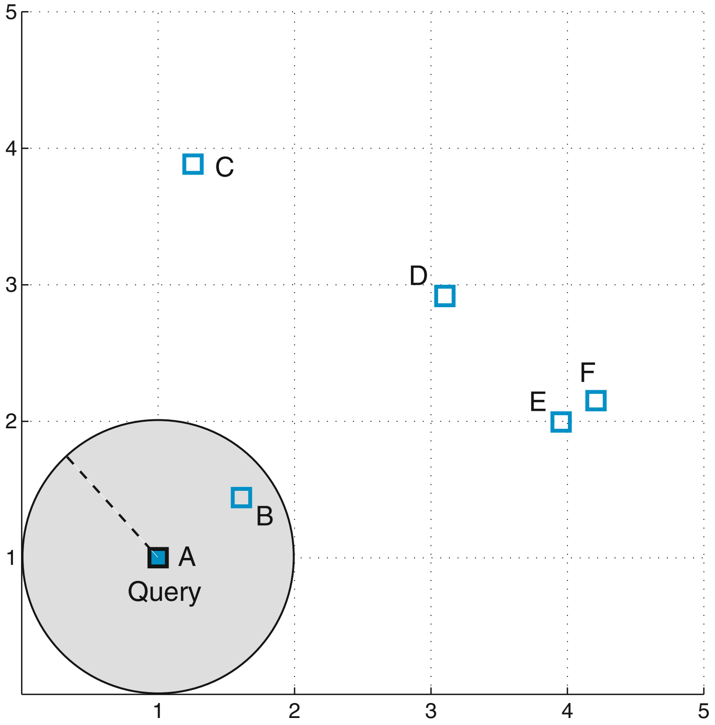
$$\text{then}$$

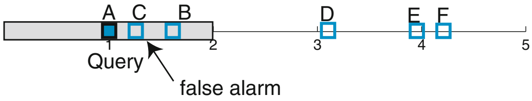
$$\Delta(q, a) \leq \Delta(Q, A)$$

As the computed distance Δ between any two compressed objects is underestimated, *false alarms* may arise. Suppose, for example, that our database consists of 6 two-dimensional points (Fig. 7). If the user query is: “Find everything that lies within a radius of 1 around A,” then B is the only result.

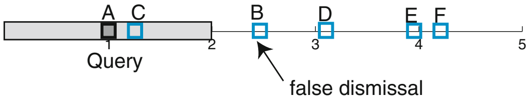
Let us assume for a minute that the dimensionality reduction performed on the data is simply a projection on the x-axis (Fig. 8). In this new space, seeking for points within a range of 1 from A would also retrieve point C, which is called a *false alarm*. However, this does not constitute a problem; in a post-processing, in a post-processing phase, the calculation of the exact

Dimensionality Reduction, Fig. 7 Range search in the original space returns only object *B*





Dimensionality Reduction, Fig. 8 Because of the dimensionality reduction, false alarms may arise



Dimensionality Reduction, Fig. 9 False dismissals may happen when the lower-bounding lemma is not obeyed

distance will remove any false alarms. Suppose now that another dimensionality reduction results in the projection of Fig. 9. Here, we have a case of a *false dismissal*, because object *B* lies outside the range of search.

This generic framework for similarity search using dimensionality reduction and lower-bounding distance functions was proposed in Agrawal et al. (1993) and is called GEMINI (**GE**neric **M**ultimedia **IN**dex**I**ng). One can show that orthonormal dimensionality reduction techniques (PCA, Fourier, wavelets) satisfy the lower-bounding lemma when the distance used is the Euclidean distance.

In conclusion, by using dimensionality reduction for search operations, one can first examine the compressed objects and eliminate many of the uncompressed objects from examination by using a lower-bounding approximation of the distance function. This initial search will return a superset of the correct answers (no false dismissals). False alarms can be filtered out by computing the original distance between the remaining uncompressed objects and the query. Therefore, a significant speedup is achieved by examining only a small subset of the original raw data.

Cross-References

▶ Curse of Dimensionality

Recommended Reading

Agrawal R, Faloutsos C, Swami A (1993) Efficient similarity search in sequence databases. In: Proceedings of the foundations of data organization and algorithms, Chicago, pp 69–84

Belkin M, Niyogi P (2002) Laplacian eigenmaps and spectral techniques for embedding and clustering. Adv Neural Inf Process Syst 1:585–591

Jolliffee IT (2002) Principal component analysis. 2nd edn. Springer, New York

Keogh E, Chakrabarti K, Pazzani M, Mehrotra S (2001) Locally adaptive dimensionality reduction for indexing large time series databases. In: Proceedings of ACM SIGMOD, Santa Barbara, pp 151–162

Lin J, Keogh E, Lonardi S, Chiu B (2003) A symbolic representation of time series, with implications for streaming algorithms. In: Proceedings of the 8th ACM SIGMOD workshop on research issues in data mining and knowledge discovery, San Diego

Roweis S, Saul L (2000) Nonlinear dimensionality reduction by locally linear embedding. Science 290(5500):2323–2326

van der Maaten LJP, Postma EO, van den Herik HJ (2009) Dimensionality reduction: a comparative review. Technical report, Tilburg University, TiCC-TR 2009-005

Tenenbaum JB, de Silva V, Langford JC (2000) A global geometric framework for nonlinear dimensionality reduction. Science 290(5500): 2319–2323

Dimensionality Reduction on Text via Feature Selection

▶ Feature Selection in Text Mining

Directed Graphs

▶ Digraphs

Dirichlet Process

Yee Whye Teh
University College London, London, UK

Definition

The Dirichlet process (DP) is a stochastic process used in ▶ Bayesian nonparametric models of data, particularly in Dirichlet process mixture models (also known as infinite mixture models). It is a distribution over distributions, that is, each draw from a Dirichlet process is itself a distribution. It is called a Dirichlet process because it has Dirichlet distributed finite dimensional marginal distributions, just as the ▶ Gaussian process, another popular stochastic process used for Bayesian nonparametric regression, has Gaussian distributed finite dimensional marginal distributions. Distributions drawn from a Dirichlet process are discrete, but cannot be described using a finite number of parameters, thus the classification as a nonparametric model.

Motivation and Background

Probabilistic models are used throughout machine learning to model distributions over observed data. Traditional parametric models using a fixed and finite number of parameters can suffer from over- or under-fitting of data when there is a misfit between the complexity of the model (often expressed in terms of the number of parameters) and the amount of data available. As a result, model selection, or the choice of a model with the right complexity, is often an important issue in parametric modeling. Unfortunately, model selection is an operation that is fraught with difficulties, whether we use ▶ cross validation or marginal probabilities as the basis for selection. The Bayesian nonparametric approach is an alternative to parametric modeling and selection. By using a model with an unbounded complexity, underfitting is mitigated, while the Bayesian approach of computing or approximating the full posterior over parameters mitigates overfitting. For a general overview of Bayesian nonparametrics, see ▶ Bayesian Nonparametric Models.

Nonparametric models are also motivated philosophically by Bayesian modeling. Typically we assume that we have an underlying and unknown distribution which we wish to infer given some observed data. Say we observe $x_1, \ldots, x_n$, with $x_i \sim F$ independent and identical draws from the unknown distribution F. A Bayesian would approach this problem by placing a prior over F then computing the posterior over F given data. Traditionally, this prior over distributions is given by a parametric family. But constraining distributions to lie within parametric families limits the scope and type of inferences that can be made. The nonparametric approach instead uses a prior over distributions with wide support, typically the support being the space of all distributions. Given such a large space over which we make our inferences, it is important that posterior computations are tractable.

The Dirichlet process is currently one of the most popular Bayesian nonparametric models. It was first formalized in Ferguson (1973) for general Bayesian statistical modeling, as a prior over distributions with wide support yet tractable posteriors. (Note however that related models in population genetics date back to Ewens 1972). Unfortunately the Dirichlet process is limited by the fact that draws from it are discrete distributions, and generalizations to more general priors did not have tractable posterior inference until the development of MCMC (▶ Markov chain Monte Carlo) techniques (Escobar and West 1995; Neal 2000). Since then there has been significant developments in terms of inference algorithms, extensions, theory and applications. In the machine learning, community work on Dirichlet processes date back to Neal (1992) and Rasmussen (2000).

Theory

The Dirichlet process (DP) is a stochastic process whose sample paths are probability measures with probability one. Stochastic processes are distributions over function spaces, with sample paths being random functions drawn from the distribution. In the case of the DP, it is a distribution over probability measures, which are functions with certain special properties, which allow them to be interpreted as distributions over some probability space Θ. Thus draws from a DP can be interpreted as random distributions. For a distribution over probability measures to be a DP, its marginal distributions have to take on a specific form which we shall give below. We assume that the user is familiar with a modicum of measure theory and Dirichlet distributions.

Before we proceed to the formal definition, we will first give an intuitive explanation of the DP as an infinite dimensional generalization of Dirichlet distributions. Consider a Bayesian mixture model consisting of K components:

$$\begin{aligned} \pi|\alpha &\sim Dir\left(\tfrac{\alpha}{K}, \ldots, \tfrac{\alpha}{K}\right) & \qquad \theta_k^*|H &\sim H \\ z_i|\pi &\sim Mult(\pi) & \qquad x_i|z_i, \{\theta_k^*\} &\sim F\left(\theta_{z_i}^*\right) \end{aligned} \tag{1}$$

where π is the mixing proportion, α is the pseudocount hyperparameter of the Dirichlet prior, H is the prior distribution over component parameters θ_k^*, and $F(\theta)$ is the component distribution parametrized by θ. It can be shown that for large K, because of the particular way we parametrized the Dirichlet prior over π, the number of components typically used to model n data items becomes independent of K and is approximately $O(\alpha \log n)$. This implies that the mixture model stays well defined as $K \to \infty$, leading to what is known as an infinite mixture model (Neal 1992; Rasmussen 2000). This model was first proposed as a way to sidestep the difficult problem of determining the number of components in a mixture, and as a nonparametric alternative to finite mixtures whose size can grow naturally with the number of data items. The more modern definition of this model uses a DP and with the resulting model called a DP mixture model. The DP itself appears as the $K \to \infty$ limit of the random discrete probability measure $\sum_{k=1}^{K} \pi_k \delta_{\theta_k^*}$, where δ_θ is a point mass centered at θ. We will return to the DP mixture toward the end of this entry.

Dirichlet Process

For a random distribution G to be distributed according to a DP, its marginal distributions have to be Dirichlet distributed (Ferguson 1973). Specifically, let H be a distribution over Θ and α be a positive real number. Then for any finite measurable partition $A_1, \ldots, A_r$ of Θ the vector $(G(A_1), \ldots, G(A_r))$ is random since G is random. We say G is Dirichlet process distributed with base distribution H and concentration parameter α, written $G \sim DP(\alpha, H)$, if

$$(G(A_1), \ldots, G(A_r)) \sim Dir(\alpha H(A_1), \ldots, \alpha H(A_r)) \qquad (2)$$

for every finite measurable partition $A_1, \ldots, A_r$ of Θ.

The parameters H and α play intuitive roles in the definition of the DP. The base distribution is basically the mean of the DP: for any measurable set $A \subset \Theta$, we have $E[G(A)] = H(A)$. On the other hand, the concentration parameter can be understood as an inverse variance: $V[G(A)] = H(A)(1 - H(A))/(\alpha + 1)$. The larger α is, the smaller the variance, and the DP will concentrate more of its mass around the mean. The concentration parameter is also called the strength parameter, referring to the strength of the prior when using the DP as a nonparametric prior over distributions in a Bayesian nonparametric model, and the mass parameter, as this prior strength can be measured in units of sample size (or mass) of observations. Also, notice that α and H only appear as their product in the definition (3) of the DP. Some authors thus treat $\widetilde{H} = \alpha H$, as the single (positive measure) parameter of the DP, writing $DP(\widetilde{H})$ instead of $DP(\alpha, H)$. This parametrization can be notationally convenient, but loses the distinct roles α and H play in describing the DP.

Since α describes the concentration of mass around the mean of the DP, as $\alpha \to \infty$, we will have $G(A) \to H(A)$ for any measurable A, that is $G \to H$ weakly or pointwise. However this not equivalent to saying that $G \to H$. As we shall see later, draws from a DP will be discrete distributions with probability one, even if H is smooth. Thus G and H need not even be absolutely continuous with respect to each other. This has not stopped some authors from using the DP as a nonparametric relaxation of a parametric model given by H. However, if smoothness is a concern, it is possible to extend the DP by convolving G with kernels so that the resulting random distribution has a density.

A related issue to the above is the coverage of the DP within the class of all distributions over Θ. We already noted that samples from the DP are discrete, thus the set of distributions with positive probability under the DP is small. However it turns out that this set is also large in a different sense: if the topological support of H (the smallest closed set S in Θ with $H(S) = 1$) is all of Θ, then any distribution over Θ can be approximated arbitrarily accurately in the weak or pointwise sense by a sequence of draws from $DP(\alpha, H)$. This property has consequence in the consistency of DPs discussed later.

For all but the simplest probability spaces, the number of measurable partitions in the definition (3) of the DP can be uncountably large. The natural question to ask here is whether objects satisfying such a large number of conditions as (3) can exist. There are a number of approaches to establish existence. Ferguson (1973) noted that the conditions (3) are consistent with each other, and made use of Kolmogorov's consistency theorem to show that a distribution over functions from the measurable subsets of Θ to $[0, 1]$ exists satisfying (3) for all finite measurable partitions of Θ. However it turns out that this construction does not necessarily guarantee a distribution over probability measures. Ferguson (1973) also provided a construction of the DP by normalizing a gamma process. In a later section we will see that the predictive distributions of the DP are related to the Blackwell–MacQueen urn scheme. Blackwell and MacQueen (1973) made use of this, along with de Finetti's theorem on exchangeable sequences, to prove existence of the DP. All the above methods made use of powerful and general mathematical machinery to establish existence, and often require regularity assumptions on H and Θ to apply these machinery. In a later section, we describe a stick-breaking construction of the DP due to Sethuraman (1994), which is a direct and elegant construction of the DP, which need not impose such regularity assumptions.

Posterior Distribution

Let $G \sim DP(\alpha, H)$. Since G is a (random) distribution, we can in turn draw samples from G itself. Let $\theta_1, \ldots, \theta_n$ be a sequence of independent draws from G. Note that the θ_i's take values in Θ since G is a distribution over Θ. We are interested in the posterior distribution of G given observed values of $\theta_1, \ldots, \theta_n$. Let $A_1, \ldots, A_r$ be a finite measurable partition of Θ, and let $n_k = \#\{i : \theta_i \in A_k\}$ be the number of observed values in A_k. By (3) and the conjugacy between the Dirichlet and the multinomial distributions, we have

$$\begin{aligned}&(G(A_1), \ldots, G(A_r))|\theta_1, \ldots, \theta_n\\ &\quad \sim \text{Dir}(\alpha H(A_1) + n_1, \ldots, \alpha H(A_r) + n_r)\end{aligned} \quad (3)$$

Since the above is true for all finite measurable partitions, the posterior distribution over G must be a DP as well. A little algebra shows that the posterior DP has updated concentration parameter $\alpha + n$ and base distribution $\frac{\alpha H + \sum_{i=1}^n \delta_{\theta_i}}{\alpha + n}$, where δ_i is a point mass located at θ_i and $n_k = \sum_{i=1}^n \delta_i(A_k)$. In other words, the DP provides a conjugate family of priors over distributions that is closed under posterior updates given observations. Rewriting the posterior DP, we have

$$G|\theta_1, \ldots, \theta_n \sim DP\left(\alpha + n, \frac{\alpha}{\alpha + n} H + \frac{n}{\alpha + n} \frac{\sum_{i=1}^n \delta_{\theta_i}}{n}\right) \quad (4)$$

Notice that the posterior base distribution is a weighted average between the prior base distribution H and the empirical distribution $\frac{\sum_{i=1}^n \delta_{\theta_i}}{n}$. The weight associated with the prior base distribution is proportional to α, while the empirical distribution has weight proportional to the number of observations n. Thus we can interpret α as the strength or mass associated with the prior. In the next section we will see that the posterior base distribution is also the predictive distribution of θ_{n+1} given $\theta_1, \ldots, \theta_n$. Taking $\alpha \to 0$, the prior becomes non-informative in the sense that the predictive distribution is just given by the empirical distribution. On the other hand, as the amount of observations grows large, $n \gg \alpha$, the posterior is simply dominated by the empirical distribution, which is in turn a close approximation of the true underlying distribution. This gives a consistency property of the DP: the posterior DP approaches the true underlying distribution.

Predictive Distribution and the Blackwell–MacQueen Urn Scheme

Consider again drawing $G \sim DP(\alpha, H)$, and drawing an i.i.d. (independently and identically distributed) sequence $\theta_1, \theta_2, \ldots \sim G$. Consider the predictive distribution for θ_{n+1}, conditioned on $\theta_1, \ldots, \theta_n$ and with G marginalized out. Since $\theta_{n+1}|G, \theta_1, \ldots, \theta_n \sim G$, for a measurable $A \subset \Theta$, we have

$$\begin{aligned}P(\theta_{n+1} \in A|\theta_1, \ldots, \theta_n) &= E[G(A)|\theta_1, \ldots, \theta_n]\\ &= \frac{1}{\alpha + n}\left(\alpha H(A) + \sum_{i=1}^n \delta_{\theta_i}(A)\right)\end{aligned} \quad (5)$$

where the last step follows from the posterior base distribution of G given the first n observations. Thus with G marginalized out:

$$\theta_{n+1}|\theta_1,\ldots,\theta_n \sim \frac{1}{\alpha+n}\left(\alpha H + \sum_{i=1}^{n} +\delta_{\theta_i}\right) \tag{6}$$

Therefore the posterior base distribution given $\theta_1,\ldots,\theta_n$ is also the predictive distribution of θ_{n+1}.

The sequence of predictive distributions (6) for $\theta_1,\theta_2,\ldots$ is called the Blackwell–MacQueen urn scheme (Blackwell and MacQueen 1973). The name stems from a metaphor useful in interpreting (6). Specifically, each value in Θ is a unique color, and draws $\theta \sim G$ are balls with the drawn value being the color of the ball. In addition we have an urn containing previously seen balls. In the beginning there are no balls in the urn, and we pick a color drawn from H, that is, draw $\theta_1 \sim H$, paint a ball with that color, and drop it into the urn. In subsequent steps, say the $n+1$st, we will either, with probability $\frac{\alpha}{\alpha+n}$, pick a new color (draw $\theta_{n+1} \sim H$), paint a ball with that color and drop the ball into the urn, or, with probability $\frac{n}{\alpha+n}$, reach into the urn to pick a random ball out (draw θ_{n+1} from the empirical distribution), paint a new ball with the same color, and drop both balls back into the urn.

The Blackwell–MacQueen urn scheme has been used to show the existence of the DP (Blackwell and MacQueen 1973). Starting from (6), which are perfectly well defined conditional distributions regardless of the question of the existence of DPs, we can construct a distribution over sequences $\theta_1,\theta_2,\ldots$ by iteratively drawing each θ_i given $\theta_1,\ldots,\theta_{i-1}$. For $n \geq 1$ let

$$P(\theta_1,\ldots,\theta_n) = \prod_{i=1}^{n} P(\theta_i|\theta_1,\ldots,\theta_{i-1}) \tag{7}$$

be the joint distribution over the first n observations, where the conditional distributions are given by (6). It is straightforward to verify that this random sequence is infinitely exchangeable. That is, for every n, the probability of generating $\theta_1,\ldots,\theta_n$ using (6), in that order, is equal to the probability of drawing them in any alternative order. More precisely, given any permutation σ on $1,\ldots,n$, we have

$$P(\theta_1,\ldots,\theta_n) = P(\theta_{\sigma(1)},\ldots,\theta_{\sigma(n)}) \tag{8}$$

Now de Finetti's theorem states that for any infinitely exchangeable sequence $\theta_1,\theta_2,\ldots$ there is a random distribution G such that the sequence is composed of i.i.d. draws from it:

$$P(\theta_1,\ldots,\theta_n) = \int \prod_{i=1}^{n} G(\theta_i)dP(G) \tag{9}$$

In our setting, the prior over the random distribution $P(G)$ is precisely the Dirichlet process $DP(\alpha,H)$, thus establishing existence.

A salient property of the predictive distribution (6) is that it has point masses located at the previous draws $\theta_1,\ldots,\theta_n$. A first observation is that with positive probability draws from G will take on the same value, regardless of smoothness of H. This implies that the distribution G itself has point masses. A further observation is that for a long enough sequence of draws from G, the value of any draw will be repeated by another draw, implying that G is composed only of a weighted sum of point masses, that is, it is a discrete distribution. We will see two sections below that this is indeed the case, and give a simple construction for G called the stick-breaking construction. Before that, we shall investigate the clustering property of the DP.

Clustering, Partitions, and the Chinese Restaurant Process

In addition to the discreteness property of draws from a DP, (6) also implies a ▶ clustering property. The discreteness and clustering properties of the DP play crucial roles in the use of DPs for clustering via DP mixture models, described in the application section. For now we assume that H is smooth, so that all repeated values are due to the discreteness property of the DP and not due to H itself. (Similar conclusions can be drawn when H has atoms, there is just more bookkeeping.) Since the values of draws are repeated, let $\theta_1^*,\ldots,\theta_m^*$ be the unique values among $\theta_1,\ldots,\theta_n$, and n_k be the number of repeats of θ_k^*. The predictive distribution can be equivalently written as

$$\theta_{n+1}\big|\theta_1,\ldots,\theta_n \sim \frac{1}{\alpha+n}\left(\alpha H + \sum_{k=1}^{m} n_k \delta_{\theta_k^*}\right) \tag{10}$$

Notice that value θ_k^* will be repeated by θ_{n+1} with probability proportional to n_k, the number of times it has already been observed. The larger n_k is, the higher the probability that it will grow. This is a rich-gets-richer phenomenon, where large clusters (a set of θ_i's with identical values θ_k^* being considered a cluster) grow larger faster.

We can delve further into the clustering property of the DP by looking at partitions induced by the clustering. The unique values of $\theta_1,\ldots,\theta_n$ induce a partitioning of the set $[n] = \{1,\ldots,n\}$ into clusters such that within each cluster, say cluster k, the θ_i's take on the same value θ_k^*. Given that $\theta_1,\ldots,\theta_n$ are random, this induces a random partition of $[n]$. This random partition in fact encapsulates all the properties of the DP, and is a very well-studied mathematical object in its own right, predating even the DP itself (Aldous 1985; Ewens 1972; Pitman 2002). To see how it encapsulates the DP, we simply invert the generative process. Starting from the distribution over random partitions, we can reconstruct the joint distribution (7) over $\theta_1,\ldots,\theta_n$, by first drawing a random partition on $[n]$, then for each cluster k in the partition draw a $\theta_k^* \sim H$, and finally assign $\theta_i = \theta_k^*$ for each i in cluster k. From the joint distribution (7) we can obtain the DP by appealing to de Finetti's theorem.

The distribution over partitions is called the Chinese restaurant process (CRP) due to a different metaphor. (The name was coined by Lester Dubins and Jim Pitman in the early 1980s (Aldous 1985)) In this metaphor we have a Chinese restaurant with an infinite number of tables, each of which can seat an infinite number of customers. The first customer enters the restaurant and sits at the first table. The second customer enters and decides either to sit with the first customer, or by herself at a new table. In general, the $n+1$st customer either joins an already occupied table k with probability proportional to the number n_k of customers already sitting there, or sits at a new table with probability proportional to α. Identifying customers with integers $1, 2, \ldots$ and tables as clusters, after n customers have sat down the tables define a partition of $[n]$ with the distribution over partitions being the same as the one above. The fact that most Chinese restaurants have round tables is an important aspect of the CRP. This is because it does not just define a distribution over partitions of $[n]$, it also defines a distribution over permutations of $[n]$, with each table corresponding to a cycle of the permutation. We do not need to explore this aspect further and refer the interested reader to Aldous (1985) and Pitman (2002).

This distribution over partitions first appeared in population genetics, where it was found to be a robust distribution over alleles (clusters) among gametes (observations) under simplifying assumptions on the population, and is known under the name of Ewens sampling formula (Ewens 1972). Before moving on we shall consider just one illuminating aspect, specifically the distribution of the number of clusters among n observations. Notice that for $i \geq 1$, the observation θ_i takes on a new value (thus incrementing m by one) with probability $\frac{\alpha}{\alpha+i-1}$ independently of the number of clusters among previous θ's. Thus the number of cluster m has mean and variance:

$$\begin{aligned} E[m|n] &= \sum_{i=1}^{n} \frac{\alpha}{\alpha+i-1} = \alpha(\psi(\alpha+n) - \psi(\alpha)) \\ &\simeq \log\left(1+\frac{n}{\alpha}\right) \quad \text{for} N, \alpha \gg 0, \end{aligned} \tag{11}$$

$$\begin{aligned} V[m|n] &= \alpha(\psi(\alpha+n) - \psi(\alpha)) \\ &\quad + \alpha^2(\psi'(\alpha+n) - \psi'(\alpha)) \\ &\simeq \alpha \log\left(1+\frac{n}{\alpha}\right) \quad \text{for } n > \alpha \gg 0, \end{aligned} \tag{12}$$

where $\psi(\cdot)$ is the digamma function. Note that the number of clusters grows only logarithmically in the number of observations. This slow growth of the number of clusters makes sense because of the rich-gets-richer phenomenon: we expect there to be large clusters thus the number of clusters m has to be smaller than the number of observations n. Notice that α controls the number of clusters in a direct manner, with larger α implying a larger number of clusters a priori. This intuition will help in the application of DPs to mixture models.

Stick-Breaking Construction

We have already intuited that draws from a DP are composed of a weighted sum of point masses. Sethuraman (1994) made this precise by providing a constructive definition of the DP as such, called the stick-breaking construction. This construction is also significantly more straightforward and general than previous proofs of the existence of DPs. It is simply given as follows:

$$\begin{aligned} \beta_k &\sim Beta(1,\alpha) & \theta_k^* &\sim H \\ \pi_k &= \beta_k \prod_{l=1}^{k-1} & G &= \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k^*} \end{aligned} \tag{13}$$

Then $G \sim DP(\alpha, H)$. The construction of π can be understood metaphorically as follows. Starting with a stick of length 1, we break it at β_1, assigning π_1 to be the length of stick we just broke off. Now recursively break the other portion to obtain π_2, π_3, and so forth. The stick-breaking distribution over π is sometimes written $\pi \sim GEM(\alpha)$, where the letters stand for Griffiths, Engen, and McCloskey (Pitman 2002). Because of its simplicity, the stick-breaking construction has lead to a variety of extensions as well as novel inference techniques for the Dirichlet process (Ishwaran and James 2001).

Applications

Because of its simplicity, DPs are used across a wide variety of applications of Bayesian analysis in both statistics and machine learning. The simplest and most prevalent applications include Bayesian model validation, density estimation, and clustering via mixture models. We shall briefly describe the first two classes before detailing DP mixture models.

How does one validate that a model gives a good fit to some observed data? The Bayesian approach would usually involve computing the marginal probability of the observed data under the model, and comparing this marginal probability to that for other models. If the marginal probability of the model of interest is highest we may conclude that we have a good fit. The choice of models to compare against is an issue in this approach, since it is desirable to compare against as large a class of models as possible. The Bayesian nonparametric approach gives an answer to this question: use the space of all possible distributions as our comparison class, with a prior over distributions. The DP is a popular choice for this prior, due to its simplicity, wide coverage of the class of all distributions, and recent advances in computationally efficient inference in DP models. The approach is usually to use the given parametric model as the base distribution of the DP, with the DP serving as a nonparametric relaxation around this parametric model. If the parametric model performs as well or better than the DP relaxed model, we have convincing evidence of the validity of the model.

Another application of DPs is in ▶ density estimation (Escobar and West 1995; Lo 1984; Neal 1992; Rasmussen 2000). Here we are interested in modeling the density from which a given set of observations is drawn. To avoid limiting ourselves to any parametric class, we may again use a nonparametric prior over all densities. Here again DPs are a popular. However note that distributions drawn from a DP are discrete, thus do not have densities. The solution is to smooth out draws from the DP with a kernel. Let $G \sim DP(\alpha, H)$ and let $f(x|\theta)$ be a family of densities (kernels) indexed by θ. We use the following as our nonparametric density of x:

$$p(x) = \int f(x|\theta)G(\theta)d\theta \tag{14}$$

Similarly, smoothing out DPs in this way is also useful in the nonparametric relaxation setting above. As we see below, this way of smoothing out DPs is equivalent to DP mixture models, if the data distributions $F(\theta)$ below are smooth with densities given by $f(x|\theta)$.

Dirichlet Process Mixture Models

The most common application of the Dirichlet process is in clustering data using mixture models (Escobar and West 1995; Lo 1984; Neal 1992; Rasmussen 2000). Here the nonparametric nature of the Dirichlet process translates to mixture models with a countably infinite number of components. We model a set of observations $\{x_1, \ldots, x_n\}$ using a set of latent parameters $\{\theta_1, \ldots, \theta_n\}$. Each θ_i is drawn independently and identically from G, while each x_i has distribution $F(\theta_i)$ parametrized by θ_i:

$$\begin{aligned} x_i|\theta_i &\sim F(\theta_i) \\ \theta_i|G &\sim G \\ G|\alpha, H &\sim DP(\alpha, H) \end{aligned} \tag{15}$$

Because G is discrete, multiple θ_i's can take on the same value simultaneously, and the above model can be seen as a mixture model, where x_i's with the same value of θ_i belong to the same cluster. The mixture perspective can be made more in agreement with the usual representation of mixture models using the stick-breaking construction (13). Let z_i be a cluster assignment variable, which takes on value k with probability π_k. Then (15) can be equivalently expressed as

$$\begin{array}{ll} \pi|\alpha \sim GEM(\alpha) & \theta_k^*|H \sim H \\ z_i|\pi \sim Mult(\pi) & x_i|z_i\{\theta_k^*\} \sim F(\theta_{z_i}^*) \end{array} \tag{16}$$

with $G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k^*}$ and $\theta_i = \theta_{z_i}^*$. In mixture modeling terminology, π is the mixing proportion, θ_k^* are the cluster parameters, $F(\theta_k^*)$ is the distribution over data in cluster k, and H the prior over cluster parameters.

The DP mixture model is an *infinite* mixture model – a mixture model with a countably infinite number of clusters. However, because the π_k's decrease exponentially quickly, only a small number of clusters will be used to model the data a priori (in fact, as we saw previously, the expected number of components used a priori is logarithmic in the number of observations). This is different than a finite mixture model, which uses a fixed number of clusters to model the data. In the DP mixture model, the actual number of clusters used to model data is not fixed, and can be automatically inferred from data using the usual Bayesian posterior inference framework (see Neal (2000) for a survey of MCMC inference procedures for DP mixture models). The equivalent operation for finite mixture models would be model averaging or model selection for the appropriate number of components, an approach that is fraught with difficulties. Thus infinite mixture models as exemplified by DP mixture models provide a compelling alternative to the traditional finite mixture model paradigm.

Generalizations and Extensions

The DP is the canonical distribution over probability measures and a wide range of generalizations have been proposed in the literature. First and foremost is the *Pitman–Yor process* (Ishwaran and James 2001; Pitman and Yor 1997), which has recently seen successful applications modeling data exhibiting power-law properties (Goldwater 2006; Teh 2006). The Pitman–Yor process includes a third parameter $d \in [0, 1)$, with $d = 0$ reducing to the DP. The various representations of the DP, including the Chinese restaurant process and the stick-breaking construction, have analogues for the Pitman–Yor process. Other generalizations of the DP are obtained by generalizing one of its representations. These include Pólya trees, normalized random measure, Poisson–Kingman models, species sampling models and stick-breaking priors.

The DP has also been used in more complex models involving more than one random probability measure. For example, in nonparametric regression we might have one probability measure for each value of a covariate, and in multitask settings each task might be associated with a probability measure with dependence across tasks implemented using a hierarchical Bayesian model. In the first situation, the class of models is typically called dependent Dirichlet processes (MacEachern 1999), while in the second the appropriate model is a hierarchical Dirichlet process (Teh et al. 2006).

Future Directions

The Dirichlet process, and Bayesian nonparametrics in general, is an active area of research within both machine learning and statistics. Current research trends span a number of directions. Firstly, there is the issue of efficient inference in DP models. Reference Neal (2000) is an excellent survey of the state-of-the-art in 2000, with all algorithms based on Gibbs sampling or small-step Metropolis–Hastings MCMC sampling. Since then there has been much work, including split-and-merge and large-step auxiliary variable MCMC sampling, sequential Monte Carlo, expectation propagation, and variational methods. Secondly, there has been interest in extending the DP, both in terms of new random distributions, as well as novel classes of nonparametric objects inspired by the DP. Thirdly, theoretical issues of convergence and consistency are being explored to provide frequentist guarantees for Bayesian nonparametric models. Finally, there are applications of such models, to clustering, transfer learning, relational learning, models of cognition, sequence learning, and regression and classification among others. We believe DPs and Bayesian nonparametrics will prove to be rich and fertile grounds for research for years to come.

Cross-References

- ▶ Bayesian Methods
- ▶ Bayesian Nonparametric Models
- ▶ Clustering
- ▶ Density Estimation
- ▶ Gaussian Process
- ▶ Prior Probability

Further Reading

In addition to the references embedded in the text above, we recommend the book (Hjort et al. 2010) on Bayesian nonparametrics.

Recommended Reading

Aldous D (1985) Exchangeability and related topics. In: École d’Été de Probabilités de Saint-Flour XIII-1983. Springer, Berlin, pp 1–198

Antoniak CE (1974) Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. Ann Stat 2(6):1152–1174

Blackwell D, MacQueen JB (1973) Ferguson distributions via Pólya urn schemes. Ann Stat 1:353–355

Escobar MD, West M (1995) Bayesian density estimation and inference using mixtures. J Am Stat Assoc 90:577–588

Ewens WJ (1972) The sampling theory of selectively neutral alleles. Theor Popul Biol 3:87–112

Ferguson TS (1973) A Bayesian analysis of some nonparametric problems. Ann Stat 1(2):209–230

Goldwater S, Griffiths TL, Johnson M (2006) Interpolating between types and tokens by estimating power-law generators. Adv Neural Inf Process Syst 18:459–466

Hjort N, Holmes C, Müller P, Walker S (eds) (2010) Bayesian nonparametrics. Cambridge series in statistical and probabilistic mathematics, vol 28. Cambridge University Press, Cambridge/New York

Ishwaran H, James LF (2001) Gibbs sampling methods for stick-breaking priors. J Am Stat Assoc 96(453): 161–173

Lo AY (1984) On a class of Bayesian nonparametric estimates: I. Density estimates. Ann Stat 12(1): 351–357

MacEachern S (1999) Dependent nonparametric processes. In: Proceedings of the section on Bayesian statistical science. American Statistical Association, Alexandria

Neal RM (1992) Bayesian mixture modeling. In: Proceedings of the workshop on maximum entropy and Bayesian methods of statistical analysis, vol 11. Kluwer Academic Publishers, Dordrecht/Boston, pp 197–211

Neal RM (2000) Markov chain sampling methods for Dirichlet process mixture models. J Comput Graph Stat 9:249–265

Pitman J (2002) Combinatorial stochastic processes (Technical Report 621). Department of Statistics, University of California at Berkeley. Lecture notes for St. Flour Summer School

Pitman J, Yor M (1997) The two-parameter Poisson–Dirichlet distribution derived from a stable subordinator. Ann Probab 25:855–900

Rasmussen CE (2000) The infinite Gaussian mixture model. Adv Neural Inf Process Syst 12:554–560

Sethuraman J (1994) A constructive definition of Dirichlet priors. Stat Sin 4:639–650

Teh YW (2006) A hierarchical Bayesian language model based on Pitman–Yor processes. In: Proceedings of the 21st international conference on computational linguistics and 44th annual meeting of the

association for computational linguistics, Sydney, pp 985–992
Teh YW, Jordan MI, Beal MJ, Blei DM (2006) Hierarchical Dirichlet processes. J Am Stat Assoc 101(476):1566–1581

Discrete Attribute

A **discrete attribute** assumes values that can be counted. The attribute cannot assume all values on the number line within its value range. See ▶ Attribute and ▶ Measurement Scales.

Discretization

Ying Yang
Australian Taxation Office, Box Hill, VIC,
Australia

Synonyms

Binning

Definition

Discretization is a process that transforms a ▶ numeric attribute into a ▶ categorical attribute. Under discretization, a new categorical attribute X' is formed from and replaces an existing numeric attribute X. Each value x' of X' corresponds to an interval $(a,b]$ of X. Any original numeric value x of X that belongs to $(a,b]$ is replaced by x'. The boundary values of formed intervals are often called "cut points."

Motivation and Background

Many learning systems require categorical data, while many data are numeric. Discretization allows numeric data to be transformed into categorical form suited to processing by such systems. Further, in some cases effective discretization can improve either computational or prediction performance relative to learning from the original numeric data.

Taxonomy

The following taxonomy identifies many key dimensions along which alternative discretization techniques can be distinguished.

Supervised vs. **Unsupervised** (Dougherty et al. 1995). Supervised methods use the class information of the training instances to select discretization cut points. Methods that do not use the class information are unsupervised.

Global vs. **Local** (Dougherty et al. 1995). Global methods discretize with respect to the whole training data space. They perform discretization only once, using a single set of intervals throughout a single classification task. Local methods allow different sets of intervals to be formed for a single attribute, each set being applied in a different classification context. For example, different discretizations of a single attribute might be applied at different nodes of a decision tree (Quinlan 1993).

Eager vs. **Lazy** (Hsu et al. 2000). Eager methods perform discretization prior to classification time. Lazy methods perform discretization during the process of classification.

Disjoint vs. **Nondisjoint** (Yang and Webb 2002). Disjoint methods discretize the value range of a numeric attribute into disjoint intervals. No intervals overlap. Nondisjoint methods discretize the value range into intervals that can overlap.

Parameterized vs. **Unparameterized**. Parameterized discretization requires input from the user, such as the maximum number of discretized intervals. Unparameterized discretization uses information only from data and does not need input from the user, for instance, the entropy minimization discretization (Fayyad and Irani 1993).

Univariate vs. **Multivariate** (Bay 2000). Methods that discretize each attribute in isolation are univariate. Methods that take into consideration relationships among attributes during discretization are multivariate.

Split vs. **Merge** (Kerber 1992) vs. **Single-scan** (Yang and Webb 2001). Split discretization initially has the whole value range as an interval and then continues splitting it into subintervals until some threshold is met. Merge discretization initially puts each value into an interval and then continues merging adjacent intervals until some threshold is met. Single-scan discretization uses neither split nor merge process. Instead, it scans the ordered values only once, sequentially forming the intervals.

Recommended Reading

Bay SD (2000) Multivariate discretization of continuous variables for set mining. In: Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining, pp 315–319

Dougherty J, Kohavi R, Sahami M (1995) Supervised and unsupervised discretization of continuous features. In: Proceedings of the twelfth international conference on machine learning, pp 194–202

Fayyad UM, Irani KB (1993) Multi-interval discretization of continuous-valued attributes for classification learning. In: Proceedings of the thirteenth international joint conference on artificial intelligence, pp 1022–1027

Hsu CN, Huang HJ, Wong TT (2000) Why discretization works for naïve Bayesian classifiers. In: Proceedings of the seventeenth international conference on machine learning, pp 309–406

Kerber R (1992) ChiMerge: discretization for numeric attributes. In: AAAI national conference on artificial intelligence, pp 123–128

Kononenko I (1992) Naive Bayesian classifier and continuous Attributes. Informatica 16(1):1–8

Quinlan JR (1993) C4.5: Programs for machine learning. Morgan Kaufmann Publishers, San Francisco

Yang Y, Webb G (2001) Proportional k-interval discretization for naive-Bayes classifiers. In: Proceedings of the twelfth European conference on machine learning, pp 564–575

Yang Y, Webb G (2002) Non-disjoint discretization for naive-Bayes classifiers. In: Proceedings of the nineteenth international conference on machine learning, pp 666–673

Discriminative Learning

Definition

Discriminative learning refers to any ▶ classification learning process that classifies by using a model or estimate of the probability $P(y|\mathbf{x})$ without reference to an explicit estimate of any of $P(\mathbf{x})$, $P(y, \mathbf{x})$, or $P(\mathbf{x}|y)$, where y is a class and $\mathbf{x}$ is a description of an object to be classified. Discriminative learning contrasts to ▶ generative learning which classifies by using an estimate of the joint probability $P(y, \mathbf{x})$ or of the prior probability P(y) and the conditional probability $P(\mathbf{x}|y)$.

It is also common to categorize as discriminative any approaches that are directly based on a decision risk function (such as ▶ Support Vector Machines, ▶ Artificial Neural Networks, and ▶ Decision Trees), where the decision risk is minimized without estimation of $P(\mathbf{x})$, $P(y, \mathbf{x})$, or $P(\mathbf{x}|y)$.

Cross-References

▶ Generative and Discriminative Learning

Disjunctive Normal Form

Bernhard Pfahringer
University of Waikato, Hamilton, New Zealand

Disjunctive normal form is an important normal form for propositional logic. A logic formula is in disjunctive normal form if it is a single disjunction of conjunctions of (possibly negated) literals. No more nesting and no other negations are allowed. Examples are:

$$a$$

$$\neg b$$

$$a \vee b$$

$$(a \wedge \neg b) \vee (c \wedge d)$$

$$\neg a \vee (b \wedge \neg c \wedge d) \vee (a \wedge \neg d)$$

Any arbitrary formula in propositional logic can be transformed into disjunctive normal form by application of the laws of distribution, De Morgan's laws, and by removing double negations. It is important to note that this process can lead to exponentially larger formulas which implies that the process in the worst case runs in exponential time. An example for this behavior is the following formula given in ▶ conjunctive normal form (CNF), which is linear in the number of propositional variables in this form. When transformed into disjunctive normal form (DNF), its size is exponentially larger.

$$\text{CNF: } (a_0 \vee a_1) \wedge (a_2 \vee a_3) \wedge \cdots \wedge (a_{2n} \vee a_{2n+1})$$

$$\text{DNF: } (a_0 \wedge a_2 \wedge \cdots \wedge a_{2n}) \vee (a_1 \wedge a_2 \wedge \cdots \wedge a_{2n}) \vee \cdots \vee (a_1 \wedge a_3 \wedge \cdots \wedge a_{2n+1})$$

Recommended Reading

Mendelson E (1997) Introduction to mathematical logic, 4th edn. Chapman & Hall, Princeton, p 30

Distance

▶ Similarity Measures

Distance Functions

▶ Similarity Measures

Distance Measures

▶ Similarity Measures

Distance Metrics

▶ Similarity Measures

Distribution-Free Learning

▶ PAC Learning

Divide-and-Conquer Learning

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt, Darmstadt, Deutschland
Department of Information Technology, University of Leoben, Leoben, Austria

Synonyms

Recursive partitioning; TDIDT strategy

Definition

The *divide-and-conquer* strategy is a learning algorithm for inducing ▶ Decision Trees. Its name reflects its key idea, which is to successively partition the dataset into smaller sets (the *divide* part) and recursively call itself on each subset (the *conquer* part). It should not be confused with the *separate-and-conquer* strategy which is used in the ▶ Covering Algorithm for rule learning.

Cross-References

▶ Covering Algorithm
▶ Decision Tree

Document Categorization

▶ Document Classification

Document Classification

Dunja Mladenić[1], Janez Brank[2], and
Marko Grobelnik[2]
[1]Artificial Intelligence Laboratory, Jožef Stefan Insitute, Ljubljana, Slovenia
[2]Jožef Stefan Institute, Ljubljana, Slovenia

Abstract

Document Classification analogous to general classification of instances, deals with

assigning labels to documents. The documents can be written in different natural languages, can be of different length and structure, can be written by different authors using variety of writing styles. Moreover, the documents to classify can be obtained from different sources including official news, internal company documentation, as well as public Web pages and texts from social media. In document classification, in addition to the algorithm we are using for constructing a classifier, data representation is crucial. Commonly used is word vector representation, where either raw data in the form of words and phrases is used or a more abstract form is constructed. Deep learning has been shown very effective for learning text representation using various deep learning architectures.

Synonyms

Document categorization; Supervised learning on text data

Definition

Document classification refers to a process of assigning one or more ▶ labels for a document from a predefined set of labels (also referred to class values). The main issues in document classification are connected to classification of free text giving document content, for instance, classifying Web documents on the content topic as being about arts, education, science, etc., or on the page type (personal homepage, company page, etc.), classifying news articles by their topic (politics, technology, science, health, etc.), and classifying movie reviews by their opinion (positive review, negative review). In general, one can consider different properties of a document in document classification and combine them, such as document type, authors, links to other documents, content, etc. Machine ▶ learning methods applied to document classification are based on general classification methods adjusted to handle some specifics of text data.

Motivation and Background

Documents and text data provide for valuable sources of information and their growing availability in electronic form naturally led to application of different analytic methods. One of the common ways is to take a whole vocabulary of the natural language in which the text is written as a feature set, resulting in several tens of thousands of features. In a simple setting, each feature gives a count of the word occurrences in a document. In this way, text of a document is represented as a vector of numbers. The representation of a particular document contains many zeros, as most of the words from the vocabulary do not occur in a particular document. In addition to the already mentioned two common specifics of text data, having a large number of features and a sparse data representation, it was observed that frequency of words in text generally follows Zipf's law – a small subset of words occur very frequently in texts, while a large number of words occur only rarely. Document classification takes these and some other data specifics into account when developing the appropriate classification methods.

Structure of Learning System

Document classification is usually performed by representing documents as vectors of feature; usually the features are words so each document is a word vector and the representation is referred to as the "bag-of-words" or "vector space model" representation. Classifier is then built using a set of documents that have been manually classified (Cohen and Singer 1996; Mladenić and Grobelnik 2003; Sebastiani 2002; Yang 1997).

Data Representation

In the word vector representation of a document, a vector of word weights is formed taking all the words occurring in all the documents. Most researchers have used single words when representing text, but there is also research that proposes using additional information to improve classification results. For instance, the feature set might be extended with various multi-word features, e.g., n-grams (sequences of n adjacent words), loose phrases (n-grams in which word order is ignored), or phrases based on grammatical analysis (noun phrases, verb phrases, etc.). Information external to the documents might also be used if it is available, for example, when dealing with Web pages, their graph organization can be a source of additional features (e.g., features corresponding to the adjacency matrix, features based on graph vertex statistics such as degree or PageRank, or features taken from the documents that are adjacent to the current document in the Web graph).

The commonly used approach to weighting words is based on ▶ TF-IDF weights where the number of occurrences of the word in the document, referred to as term frequency (TF), is multiplied by the importance of the word with regard to the whole corpus (▶ (IDF) inverse document frequency). The IDF weight for the ith word is defined as $\text{IDF}_i = \log(N/\text{DF}_i)$, where N is total number of documents and DF_i is the document frequency of the ith word (the number of documents from the whole corpus in which the ith word appears). The IDF weight decreases the influence of common words (which are not as likely to be useful for discriminating between classes of documents) and favors the less common words. However, the least frequently occurring words are often deleted from the documents as a preprocessing step, based on the notion that if a word that does not occur often enough in the training set cannot be useful for learning and generalization and would effectively be perceived as noise by the learning ▶ algorithm. A stopword list is also often used to delete some of the most common and low-content words (such as "the," "of," "in," etc.) during preprocessing. For many purposes, the vectors used to represent documents should be normalized to unit length so that the vector reflects the contents and themes of the document but not its length (which is typically not relevant for the purposes of document categorization).

Even in a corpus of just a few thousand documents, this approach to document representation can easily lead to a feature space of thousands, possibly tens of thousands, of features. Therefore, feature selection is sometimes used to reduce the feature set before training. Such questions as whether feature selection is needed and/or beneficial, and which feature selection method should be used, depend considerably on the learning algorithm used; the number of features to be retained depends both on the learning algorithm and on the feature selection method used. For example, ▶ naive Bayes tends to benefit, indeed require, heavy feature selection, while ▶ support vector machines (SVMs) tend to benefit little or nothing from it. Similarly, odds ratio tends to value (some) rare features highly and therefore requires a lot of features to be kept, while information gain tends to score some of the more frequent features highly and thus often works better if a smaller number of features is kept (see also ▶ Feature Selection in Text Mining).

Due to the large number of features in the original data representation, some of the more computationally expensive feature selection methods from traditional machine learning cannot be used with textual data. Typically, simple feature scoring measures, such as information gain, odds ratio, and chi-squared, are used to rank the features, and the features whose score falls below a certain threshold are discarded. A better but computationally more expensive feature scoring method is to train a linear classifier on the full feature set first (e.g., using linear ▶ SVM, see below) and rank the features by the absolute value of their weights in the resulting linear model (see also ▶ Feature Construction in Text Mining).

Classification

Different ▶ classification algorithms have been adjusted and applied on text data. A few more popular are described here.

▶ Naive Bayes based on the multinomial model, where the predicted class for document d is the one that maximizes the ▶ posterior probability $P(c|d) \propto P(c)\Pi_t P(t|c)$ TF(t, d), where $P(c)$ is the ► prior probability that a document belongs to class c, $P(t|c)$ is the probability that a word chosen randomly in a document from class c equals t, and TF(t, d) is the "term frequency," or the number of occurrences of word t in a document d. Where there are only two classes, say c_+ and c_-, maximizing $P(c|d)$ is equivalent to taking the sign of ln $P(c_+|d)/P(c_-|d)$, which is a linear combination of TF(w, d). Thus, the naive Bayes classifier can be seen as a linear classifier as well. The training consists simply of estimating the probabilities $P(t|c)$ and $P(c)$ from the training documents.

► Perceptron trains a linear classifier in an incremental way as a neural unit using an additive update rule. The prediction for a document represented by the vector $\mathbf{x}$ is sgn($\mathbf{w}^T$ $\mathbf{x}$), where $\mathbf{w}$ is a vector of weights obtained during training. Computation starts with $\mathbf{w} = 0$ and then considers each training example $\mathbf{x}_i$ in turn. If the present $\mathbf{w}$ classifies document $\mathbf{x}_i$ correctly, it is left unchanged; otherwise, it is updated according to the additive rule: $\mathbf{w} \leftarrow \mathbf{w} + y_i\ \mathbf{x}_i$, where y_i is the correct class label of the document $\mathbf{x}_i$, namely, $y_i = +1$ for a positive document and $y_i = 1$ for a negative one.

► SVM trains a linear classifier of the form sgn ($\mathbf{w}^T$ $\mathbf{x}$ $+b$). Learning is posed as an optimization problem with the goal of maximizing the *margin*, i.e., the distance between the separating hyperplane $\mathbf{w}^T\ \mathbf{x} + b = 0$ and the nearest training vectors. An extension of this formulation, known as the *soft margin*, also allows for a wider margin at the cost of misclassifying some of the ► training examples. The dual form of this optimization task is a quadratic programming problem and can be solved numerically.

Results of numerous experiments reported in research papers suggest that among the classification algorithms that have been adjusted to text data SVM, ► naive Bayes and k-nearest neighbor are among the best performing (Lewis et al. 1996). Moreover, experimental evaluation on some standard Reuters news datasets shows that SVM tends to outperform other classifiers including naive Bayes and perceptron (Mladenic et al. 2004).

In many applications, a document may belong to multiple classes, e.g., because it includes discussion relevant to several topics. The classifiers described above can naturally provide multi-label predictions in a multi-class learning problem simply by treating each class as a two-class problem separately from the other classes. However, there are also methods that try to model the multi-class nature of individual documents directly. For example, recently there has been an increase of interest in using artificial neural networks for text classification, with a multiunit output layer that can generate predictions for all classes at once (Zhang and Zhou 2006; Nam et al. 2014).

Evaluation Measures

A ► characteristic property of machine learning problems arising in document classification is a very unbalanced class distribution. In a typical dataset, there may be tens (or sometimes hundreds or thousands) of categories, most of which are very small. When we train a binary (two-class) classification model for a particular category, documents belonging to that category are treated as the positive class, while all other documents are treated as the negative class. Thus, the negative class is typically vastly larger as the positive one. These circumstances are not well suited to some traditional machine learning ► evaluation measures, such as ► accuracy (if almost all documents are negative, then a useless classifier that always predicts the negative class will have very high accuracy). Instead, evaluation measures from information retrieval are more commonly used, such as ► precision, ► recall, the F_1-measure, the ► breakeven point (BEP), and the area under the ► receiver operating characteristic (ROC) curve (see also ► ROC Analysis).

The evaluation of a binary classifier for a given category c on a given ► test set can be conveniently summarized in a contingency table. We can divide documents into four groups depending on whether they belong to c and whether our classifier predicted them as positive (i.e., supposedly belonging to c) or not:

Given the number of documents in each of the four groups (TP, FP, TN, and FN), we can compute various evaluation measures as follows:

- Precision = TP/(TP + FP)
- Recall = TP_{rate} = TP/(TP + FN)
- FP_{rate} = FP/(TN + FP)
- $F_1 = 2 \bullet precision \bullet recall/(precision + recall)$

	Belongs to *c*	Not in *c*
Predicted positive	TP (true positives)	FP (false positives)
Predicted negative	FN (false negatives)	TN (true negatives)

Thus, precision is the proportion of documents predicted positive that are really positive, while recall is the proportion of positive documents that have been correctly predicted as positive. The F_1 is the ▶ harmonic mean of precision and recall; thus, it lies between ▶ precision and recall but is closer to the lower of these two values. This means that a classifier with high F_1 has both good precision and good recall. In practice, there is usually a tradeoff between precision and recall; by making the classifier more liberal (i.e., more likely to predict positive), we can increase recall at the expense of precision, while by making it more conservative (less likely to predict positive), we can usually increase precision at the expense of recall. Often the classification model involves a threshold which can be varied at will to obtain various $\langle precision, recall\rangle$ pairs. These can be plotted on a chart, resulting in the *precision-recall curve*. As we decrease the threshold (thus making the classifier more liberal), precision decreases and recall increases until at some point precision and recall are equal; this value is known as the (*precision-recall*) *BEP* (Lewis 1991). Instead of $\langle precision, recall\rangle$ pairs, one can measure $\langle \text{TP}_{\text{rate}}, \text{FP}_{\text{rate}}\rangle$ pairs, resulting in an ▶ ROC curve (see ▶ ROC analysis). The ▶ area under the ROC curve is another valuable measure of the classifier quality.

Document classification problems are typically multi-class, multi-label problems, which are treated by regarding each category as a separate two-class classification problem. After training a two-class classifier for each category and evaluating it, the question arises how to combine these evaluation measures into an overall evaluation measure. One way is *macroaveraging*, which means that the values of precision, recall, F_1, or whatever other measure we are interested in are simply averaged over all the categories. Since small categories tend to be much more numerous than large ones, macroaveraging tends to emphasize the performance of our learning algorithm on small categories. An alternative approach is *microaveraging*, in which the contingency tables for individual two-class classifiers are summed up and measures such as precision, recall, and F_1 computed from the resulting aggregated table. This approach emphasizes the performance of our learning algorithm on larger categories.

Cross-References

▶ Classification
▶ Feature Selection
▶ Precision
▶ Semi-supervised Text Processing
▶ Support Vector Machines
▶ Text Visualization

Recommended Reading

Cohen WW, Singer Y (1996) Context sensitive learning methods for text categorization. In: Proceedings of the 19th annual international ACM SIGIR conference on research and development in information retrieval. ACM, Zurich, pp 307–315

Lewis DD (1991) Representation and learning in information retrieval. PhD thesis, Department of computer science, University of Massachusetts, Amherst

Lewis DD, Schapire RE, Callan JP, Ron Papka R (1996) Training algorithms for linear text classifiers. In: Proceedings of the 19th annual international ACM SIGIR conference on research and development in information retrieval SIGIR-1996. ACM, New York, pp 298–306

Mladenic D, Brank J, Grobelnik M, Milic-Frayling N (2004) Feature selection using linear classifier weights: interaction with classification models. In: Proceedings of the twenty-seventh annual international ACM SIGIR conference on research and development in information retrieval SIGIR-2004. ACM, New York, pp 234–241
Mladenić D, Grobelnik M (2003) Feature selection on hierarchy of Web documents. J Decis Support Syst 35:45–87
Nam J, Kim J, Mencia EL, Gurevych I, Fürnkranz J (2014) Large-scale multi-label text classification – revisiting neural networks. In: Proceedings of ECML/PKDD, Nancy, pp 437–452
Sebastiani F (2002) Machine learning for automated text categorization. ACM Comput Surv 34(1):1–47
Yang Y (1997) An evaluation of statistical approaches to text categorization. J Info Retr 1:67–88
Zhang ML, Zhou ZH (2006) Multilabel neural networks with applications to functional genomics and text categorization. IEEE Trans Knowl Data Eng 18:1338–1351

Domain Adaptation

▶ Inductive Transfer

Dual Control

▶ Bayesian Reinforcement Learning
▶ Partially Observable Markov Decision Processes

Duplicate Detection

▶ Entity Resolution

Dynamic Bayesian Network

▶ Learning Graphical Models

Dynamic Decision Networks

▶ Partially Observable Markov Decision Processes

Dynamic Programming

Martin L. Puterman[1] and Jonathan Patrick[2]
[1]University of British Columbia, Vancouver, BC, Canada
[2]University of Ottawa, Ottawa, ON, Canada

Definition

Dynamic programming is a method for modeling a sequential decision process in which past decisions impact future possibilities. Decisions can be made at fixed discrete time intervals or at random time intervals triggered by some change in the system. The decision process can last for a finite period of time or run indefinitely – depending on the application. Each time a decision needs to be made, the decision-maker (referred to as "he" in this entry with no sexist connotation intended) views the current ▶ state of the system and chooses from a known set of possible ▶ actions. As a result of the state of the system and the action chosen, the decision-maker receives a reward (or pays a ▶ cost) and the system evolves to a new state based on known probabilities. The challenge faced by the decision-maker is to choose a sequence of actions that will lead to the greatest reward over the length of the decision-making horizon. To do this, he needs to consider not only the current reward (or cost) for taking a given action but the impact such an action might have on future rewards. A policy is a complete sequence of decisions that dictates what action to take in any given state and at any given time. Dynamic programming finds the optimal policy by developing mathematical recursions that decompose the multi-decision problem into a series of single-decision problems that are analytically or computationally more tractable.

Background and Motivation

The earliest concepts that later developed into dynamic programming can be traced back to the calculus of variation problems in the seventeenth century. However, the modern investigation of stochastic sequential decision problems arguably dates back to the work by Wald in 1947 on sequential statistical analysis. At much the same time, Pierre Masse was analyzing similar problems applied to water resource management in France. However, the major name associated with dynamic programming is that of Richard Bellman who established the optimality equations that form the basis of dynamic programming.

It is not hard to demonstrate the potential scope of dynamic programming. Table 1 gives a sense of the breadth of application as well as highlighting the stochastic nature of most instances.

Structure of the Learning System

A dynamic program is a general representation of a sequential decision problem under uncertainty about the future and is one of the main methods for solving Markov decision problems (see ▸ Markov Decision Processes). Like a decision tree, it models a process where the decision we make "today" impacts where we end up tomorrow and therefore what decisions are available to us tomorrow. It has distinct advantages over a decision tree in that:

- It is a more compact representation of a decision process
- It enables efficient calculation
- It allows exploration of the structural properties of optimal decisions
- It can analyze and solve problems with infinite or indefinite time horizons

The Finite-Horizon Setting

A finite-horizon MDP is a decision process with a known end date. Thus, the decision-maker is faced with the task of making a finite sequence of decisions at fixed intervals. The MDP model is based on five elements:

- ▸ Decision epochs: Sequences of decision times $n = 1, \ldots, N$ (in the infinite horizon, we set $N = \infty$). In a discrete-time MDP, these decision times happen at regular, fixed intervals while in a continuous-time model, they occur at random times triggered by a change in the system. The time between decision epochs is called a period.
- ▸ State space: States represent the possible system configurations facing the decision-maker at each decision epoch. They contain all information available to the decision-maker at each decision epoch. The state space, S, is the set of all such states (often assumed to be finite). In choosing the state space, it is important to include all the information that may be relevant in determining a decision and that may change from decision epoch to decision epoch.
- ▸ Actions: Actions are the available choices for the decision-maker at any given decision epoch, in any given state. $A(s)$ is the set of all actions available in state s (usually assumed to be finite for all s). No action is taken in the final decision epoch N.
- ▸ Transition probabilities: The probability of being in state s' at time $t+1$, given you take action a from state s at time t, is written as $p_t(s'|s,a)$. It clearly makes sense to allow the transition probabilities to be conditional upon the current state and the action taken.
- ▸ Rewards/costs: In most MDP applications, the decision-maker receives a reward each period. This reward can depend on the current state, the action taken, and the next state and is denoted by $r_t(s,a,s')$. Since a decision must be made before knowing the next state, s', the MDP formulation deals with the expected reward:

$$r_t(s,a) = \sum_{s' \in S} r_t(s,a,s') p_t(s'|s,a).$$

We also define the terminal rewards as $r_N(s)$ for being in state s at the final decision epoch.

Dynamic Programming, Table 1 Dynamic programming applications

Application	System state	Actions	Rewards	Stochastic aspect
Capacity	Size of plant	Maintain or add capacity	Costs of expansion and production at current capacity	Demand for a product
Cash mgt	Cash available	Borrow or invest	Transaction costs and less interest	External demand for cash
Catalog mailing	Customer purchase record	Type of catalog to send, if any	Purchases in current period less mailing costs	Customer purchase amount
Clinical trials	Number of successes with each treatment	Stop or continue the trial	Costs of treatment and incorrect decisions	Response of a subject to treatment
Economic growth	State of the economy	Investment or consumption	Utility of consumption	Effect of investment
Fisheries mgt	Fish stock in each age class	Number of fish to harvest	Value of the catch	Population size
Forest mgt	Size and condition of stand	Harvesting and reforestation activities	Revenues and less harvesting costs	Stand growth and price fluctuation
Gambling	Current wealth	Stop or continue playing	Cost of playing	Outcome of the game
Inventory control	Stock on hand	Order additional stock	Revenue per item sold and less ordering, holding, and penalty costs	Demand for items
Project selection	Status of each project	Project to invest in at present	Return from investing in project	Change in project status
Queueing control	Number in the queue	Accept/reject new customers or control service rate	Revenue from serving customers and less delay costs	Interarrival times and service times
Reliability	Age or status of equipment	Inspect and repair or replace if necessary	Inspection, repair, and failure costs	Failure and deterioration
Reservations	Number of confirmed reservations	Accept, wait-list, or reject new reservation	Profit from satisfied reservations and less overbooking penalties	Number of arrivals and the demand for reservations
Scheduling	Activities completed	Next activity to schedule	Cost of activity	Length of time to complete activity
Selling an asset	Current offer	Accept or reject the offer	The offer is less than the cost of holding the asset for one period	Size of the offer
Water resource management	Level of water in each reservoir	Quantity of water to release	Value of power generated	Rainfall and runoff

These are independent of the action since no action is taken at that point.

The objective in the finite-horizon model is to maximize total expected reward:

$$\max\left\{E\left[\sum_{t=1}^{N} r_t(s_t,a_t,s_{t+1})+r_N(s_N)|s_1=s\right]\right\}. \tag{1}$$

At any given time t, the decision-maker has observed the history up to time t, represented by $h_t = (s_1, a_1, s_2, a_2, \ldots, a_{t-1}, s_t)$, and needs to choose a_t in such a way as to maximize (1). A ▶ decision rule, d_t, determines what action to take, based on the history to date at a given decision epoch and for any possible state. It is deterministic if it selects a single member of $A(s)$ with probability 1 for each $s \in S$ and for a given h_t, and it is ▶ randomized (▶ randomized decision rule) if it selects a member of $A(s)$ at random with probability $q_{d_t(h_t)}(a)$. It is Markovian (▶ Markovian decision rule) if it depends on h_t only through s_t. That is, $d_t(h_t) = d_t(s_t)$.

A policy, $\pi = (d_1, \ldots, d_{N-1})$, denotes a complete sequence of decision rules over the whole horizon. It can be viewed as a "contingency plan" that determines the action for each possible state at each decision epoch. One of the major results in MDP theory is that, under reasonable conditions, it is possible to prove that there exists a Markovian, deterministic policy that attains the maximum total expected reward. Thus, for the purposes of this entry, we will concentrate on this subset of all policies.

If we define $v_t(s)$ as the expected total reward from time t to the end of the planning horizon, given that at time t the system occupies state s, then a recursion formula can be built that represents v_t in terms of v_{t+1}. Specifically,

$$v_t(s) = \max_{a \in A(s)} \left\{ r_t(s,a) + \sum_{s' \in S} p(s'|s,a) v_{t+1}(s') \right\} \tag{2}$$

This is often referred to as the ▶ Bellman equation, named after Richard Bellman who was responsible for the seminal work in this area. It breaks the total reward at time t into the immediate reward $r_t(s,a)$ and the total future expected reward, $\sum_{s' \in S} p(s'|s,a) v_{t+1}(s')$. Define $A^*_{s,t}$ as the set of actions that attain the maximum in (2) for a given state s and decision epoch t. Then the finite-horizon discrete-time MDP can be solved through the following backward induction algorithm.

Backward Induction Algorithm

- Set $t = N$ and $v_t(s) = r_N(s) \quad \forall s \in S$ (since there is no decision at epoch N and no future epochs, it follows that the optimal reward-to-go function is just the terminal reward).
- Let $t = t - 1$ and compute for each $s \in S_t$

$$v_t(s) = \max_{a \in A(s)} \left\{ r_t(s,a) + \sum_{s' \in S} p(s'|s,a) v_{t+1}(s') \right\} .$$

- For each $s \in S_t$, compute $A^*_{s,t}$ by solving

$$\text{argmax}_{a \in A(s)} \left\{ r_t(s,a) + \sum_{s' \in S} p(s'|s,a) v_{t+1}(s') \right\} .$$

- If $t = 1$ then stop else return to step 2.

The function $v_1(s)$ is the maximum expected reward over the entire planning horizon given the system starts in state s. The optimal policy is constructed by choosing a member of $A^*_{s,t}$ for each $s \in S$ and $t \in \{1, \ldots, N\}$. In essence, the algorithm solves a complex N-period decision problem by solving N simple 1-period decision problems.

Example – inventory control: Periodically (daily, weekly, or monthly), an inventory manager must determine how much of a product to stock in order to satisfy random external demand for the product. If too little is in stock, potential sales are lost. Conversely, if too much is on hand, a cost for carrying inventory is incurred. The objective is to choose an ordering rule that maximizes expected total profit (sales minus holding and ordering costs) over the planning horizon. To formulate an MDP model of this system requires precise assumptions such as:

- The decision regarding the quantity to order is made at the beginning of each period and delivery occurs instantaneously.

- Demand for the product arrives throughout the period, but all orders are filled on the last day of the period.
- If demand exceeds the stock on hand, potential sales are lost.
- The revenues, costs, and demand distribution are the same each period.
- The product can only be sold in whole units.
- The warehouse has a capacity for M units.

(These assumptions are not strictly necessary but removing them leads to a different formulation.) Decisions epochs correspond to the start of a period. The state, $s_t \in \{0, \ldots, M\}$, represents the inventory on hand at the start of period t and the action, $a_t \in \{0, 1, 2, \ldots, M - s\}$, is the number of units to order that period; the action 0 corresponds to not placing an order. Let D_t represent the random demand throughout period t and assume that the distribution of demand is given by $p_t(d) = P(D_t = d), d = 0, 1, 2, \ldots$. The cost of ordering u units is $O(u) = K + c(u)$ (a fixed cost plus variable cost) and the cost of storing u units is $h(u)$, where $c(u)$ and $h(u)$ are increasing functions in u. We will assume that leftover inventory at the end of the planning horizon has value $g(u)$ and that the sale of u units yields a revenue of $f(u)$. Thus, if there are u units on hand at decision epoch t, the expected revenue is

$$F_t(u) = \sum_{j=0}^{u-1} f(j)p_t(j) + f(u)P(D_t \geq u).$$

The expected reward is therefore

$$r_t(s, a) = F(s + a) - O(a) - h(s + a)$$

and the terminal rewards are $r_N(s, a) = g(s)$. Finally, the transition probabilities depend on whether or not there is enough stock on hand, $s + a$, to meet the demand for that month, D_t. Specifically,

$$p_t(j|s, a) = \begin{cases} 0 & \text{if } j > s + a, \\ p_t(j) & \text{if } j = s + a - D_t, s + a \leq M, \\ & \quad s + a > D_t, \\ \sum_{d=s+a}^{\infty} p_t(d) & \text{if } j = 0, s + a \leq M, s + a \leq D_t. \end{cases}$$

Solving the finite-horizon version of this problem through backward induction reveals a simple form to the optimal policy referred to as an (s, S) policy. Specifically, if at time t, the inventory is below some number s^t, then it is optimal to order a quantity that raises the inventory level to S^t. It has been shown that a structured policy of this type is optimal for several variants of the inventory management problem with a fixed ordering cost. Many variants of this problem have been studied; these models underlie the field of supply chain management.

The Infinite-Horizon Setting

In the infinite (or indefinite)-horizon setting, the backward induction algorithm described above no longer suffices as there are no terminal rewards with which to begin the process.

In most finite-horizon problems, the optimal policy begins to look the same at each decision epoch as the horizon is pushed further and further into the future. For instance, in the inventory example above, $s^t = s^{t+1}$ and $S^t = S^{t+1}$ if t is sufficiently removed from the end of the horizon. The form of the optimal policy only changes as the end of the time horizon approaches. Thus, if there is no fixed time horizon, we should expect the optimal policy to be *stationary* in most cases. We call a policy *stationary* if the same decision rule is applied at each decision epoch (i.e., $d_t = d \; \forall \, t$). One necessary assumption for this to be true is that the rewards and transition probabilities are independent of time (i.e., $r_t(s, a) = r(s, a)$ and $p_t(s'|s, a) = p(s'|s, a) \, \forall \, s,' s \in S$ and $a \in A(s)$). For the infinite-horizon MDP, the theory again proves that under mild assumptions, there exists an optimal policy that is *stationary, deterministic*, and *Markovian*. This fact greatly simplifies the process of finding the optimal policy as we can concentrate on a small subset of all potential policies.

The setup for the infinite-horizon MDP is entirely analogous to the finite-horizon setting with the same ▶ decision epochs, ▶ states, ▶ actions, ▶ rewards, and ▶ transition probabilities (with the last two assumed to be independent of time).

The most obvious objective is to extend the finite-horizon objective to infinity and seek to find the policy, π, that maximizes the total expected reward:

$$v^{\pi}(s) = \lim_{N\to\infty}\left\{E_s^{\pi}\left[\sum_{t=1}^{N} r(s_t, a_t)\right]\right\} . \qquad (3)$$

This, however, is problematic since

1. The sum may be infinite for some or all policies
2. The sum may not even exist, or
3. Even if the sum exists, there may be no maximizing policy

In the first case, just because all (or a subset of all) policies lead to infinite reward in the long run does not mean that they are all equally beneficial. For instance, one may give a reward of $100 each epoch and the other $1 per epoch. Alternatively, one may give large rewards earlier on while another gives large rewards only much later. Generally speaking, the first is more appealing but the above objective function will not differentiate between them. Secondly, the limit may not exist if, for instance, the reward each decision epoch oscillates between 1 and -1. Thirdly, there may be no maximizing policy simply because there is an infinite number of policies and thus there may be an infinite sequence of policies that converges to a maximum limit but never reaches it. Thus, instead we look to maximize either the *total expected discounted reward* or the *expected long-run average reward* depending on the application.

Let $\lambda \in (0,1)$ be a discount factor. Assuming the rewards are bounded (i.e., there exists an M such that $|r(s,a)| < M \quad \forall (s,a) \in S \times A(s)$), the *total expected discounted reward* for a given policy π is defined as

$$\begin{aligned} v_{\lambda}^{\pi}(s) &= \lim_{N\to\infty} E_s^{\pi}\left\{\sum_{t=1}^{N} \lambda^{t-1} r(s_t, d_t(s_t))\right\} \\ &= E_s^{\pi}\left\{\sum_{t=1}^{\infty} \lambda^{t-1} r(s_t, d_t(s_t))\right\} . \end{aligned}$$

Since $\lambda < 1$ and the rewards are bounded, this limit always exists. The second objective is the *expected average reward* which, for a given policy π, is defined as

$$g^{\pi}(s) = \lim_{N\to\infty} \frac{1}{N} E_s^{\pi}\left\{\sum_{t=1}^{N} r(s_t, d_t(s_t))\right\} .$$

Once again, we are dealing with a limit that may or may not exist. As we will see later, whether the above limit exists depends on the structure of the Markov chain induced by the policy.

Let us, at this point, formalize what we mean by an optimal policy. Clearly, that will depend on which objective function we choose to use. We say that

- π^* is *total reward optimal* if $v^{\pi^*}(s) \geq v^{\pi}(s)$ $\forall s \in S$and $\forall \pi$.
- π^* is *discount optimal* if $v_{\lambda}^{\pi^*}(s) \geq v_{\lambda}^{\pi}(s)$ $\forall s \in S$and $\forall \pi$.
- π^* is *average optimal* if $g^{\pi^*}(s) \geq g^{\pi}(s)$ $\forall s \in S$ and $\forall \pi$.

For simplicity, we introduce matrix and vector notation. Let $r_d(s) = r(s, d(s))$ and $p_d(j|s) = p(j|s, d(s))$. Thus r_d is the vector of rewards for each state under decision rule d, and P_d is the transition matrix of states under decision rule d. We will now take a more in-depth look at the infinite-horizon model with the total expected discounted reward as the optimality criterion.

Solving the Discounted Infinite-Horizon MDP

Given a Markovian, deterministic policy $\pi = (d_1, d_2, d_3, \ldots)$ and defining $\pi_k = (d_k, d_{k+1}, \ldots)$, we can compute

$$
\begin{aligned}
v_\lambda^\pi(s) &= E_s^{\pi_1}\left[\sum_{t=1}^{\infty} \lambda^{t-1} r(s_t, d_t(s_t))\right] \\
&= E_s^{\pi_1}\left[r(s, d_1(s)) + \lambda \sum_{t=2}^{\infty} \lambda^{t-2} r(s_t, d_t(s_t))\right] \\
&= r(s, d_1(s)) + \lambda \sum_{j\in S} p_{d_1}(j|s) E_j^{\pi_2}\left[\sum_{t=1}^{\infty} \lambda^{t-1} r(s_t, d_t(s_t))\right] \\
&= r(s, d_1(s)) + \lambda \sum_{j\in S} p_{d_1}(j|s) v_\lambda^{\pi_2}(j).
\end{aligned}
$$

In matrix notation,

$$v_\lambda^{\pi_1} = r_{d_1} + \lambda P_{d_1} v_\lambda^{\pi_2}.$$

If we follow our supposition that we need to only consider *stationary* policies (so that the same decision rule is applied to every decision epoch), $\pi = d^\infty = (d, d, \ldots)$, then this results in

$$v_\lambda^{d^\infty} = r_d + \lambda P_d v_\lambda^{d^\infty}.$$

This implies that the value function generated by a stationary policy satisfies the equation:

$$
\begin{aligned}
v &= r_d + \lambda P_d v \\
\Rightarrow v &= (I - \lambda P_d)^{-1} r_d.
\end{aligned}
$$

The inverse above always exists since P_d is a probability matrix (so that its spectral radius is less than or equal to 1) and $\lambda \in (0, 1)$. Moving to the maximization problem of finding the optimal policy, we get the recursion formula

$$v(s) = \max_{a\in A(s)} \left\{ r(s,a) + \lambda \sum_{j\in S} p(s|s,a) v(j) \right\}. \tag{4}$$

Note that the right-hand side can be viewed as a function of a vector v (given r, p, λ). We define a vector-valued function

$$Lv = \max_{d\in D^{MD}} \left\{ r_d + \lambda P_d v \right\},$$

where D^{MD} is the set of all Markovian, ▶ deterministic decision rules. There are three methods for solving the above optimization problem in order to determine the optimal policy. The first method, called *value iteration*, creates a sequence of approximations to the value function that eventually converges to the value function associated with the optimal policy.

Value Iteration

1. Start with an arbitrary $|S|$-vector v^0. Let $n = 0$ and choose $\epsilon > 0$ to be small.
2. For every $s \in S$, compute $v^{n+1}(s)$ as

$$
\begin{aligned}
&v^{n+1}(s) \\
&= \max_{a\in A(s)} \left\{ r(s,a) + \sum_{j\in S} \lambda p(j|s,a) v^n(j) \right\}.
\end{aligned}
$$

3. If $\max_{s\in S} |v^{n+1}(s) - v^n(s)| \geq \epsilon(1-\lambda)/2\lambda$ let $n \to n+1$ and return to step 2.
4. For each $s \in S$, choose

$$d_\epsilon(s) \in \text{argmax}_{a\in A(s)} \left\{ r(s,a) + \sum_{j\in S} \lambda p(j|s,a) v^{n+1}(j) \right\}.$$

It has been shown that value iteration identifies a policy with expected total discounted reward within ϵ of optimality in a finite number of iterations. Many variants of value iteration are available such as using different stopping criteria to accelerate convergence or combining value iteration with the policy iteration algorithm described below.

A second algorithm, called *policy iteration*, iterates through a sequence of policies eventually converging to the optimal policy.

Policy Iteration

1. Set $d_0 \in D$ to be an arbitrary policy. Let $n = 0$.
2. (Policy evaluation) Obtain v^n by solving
$$v^n = (I - \lambda P_{d_n})^{-1} r_{d_n}.$$
3. (Policy improvement) Choose d_{n+1} to satisfy
$$d_{n+1} \in \text{argmax}_{d \in D}\{r_d + \lambda P_d v^n\}$$
componentwise. If d_n is in this set, then choose $d_{n+1} = d_n$.
4. If $d_{n+1} = d_n$, set $d^* = d_n$ and stop. Otherwise, let $n \to n+1$ and return to (2).

Note that value iteration and policy iteration have different conceptual underpinnings. Value iteration seeks a *fixed point* of the operator L using successive approximations, while policy iteration can be viewed as using Newton's method to solve $Lv - v = 0$.

Finally, a third method for solving the discounted infinite-horizon MDP takes advantage of the fact that, because L is monotone, if $Lv \leq v$, then $L^2 v \leq Lv$ and more generally, $L^k v \leq v$. Thus, induction implies that the value function of the optimal policy, v^*_λ, is less than or equal to v for any v, where $Lv \leq v$. We define the set $U := \{v \in V | Lv \leq v\}$. Then, not only is v^*_λ in the set U, it is also the smallest element of U. Therefore, we can solve for v^*_λ by solving the following linear program:

$$\min_v \sum_{s \in S} \alpha(s) v(s)$$

subject to

$$v(s) \geq r(s,a) + \lambda \sum_{j \in S} p(j|s,a) v(j) \quad \forall s \in S, \;\; a \in A_s.$$

(Note that the above set of constraints is equivalent to $Lv \leq v$.) We call this the primal LP. The coefficients $\alpha(s)$ are arbitrarily chosen. The surprising fact is that the solution to the above LP will be v^*_λ for any strictly positive α.

We can construct the dual to the above primal to get

$$\max_X \sum_{s \in S} \sum_{a \in A_s} r(s,a) X(s,a)$$

subject to

$$\sum_{a \in A_j} X(j,a) - \sum_{s \in S} \sum_{a \in A_s} \lambda p(j|s,a) X(s,a) = \alpha(j) \quad \forall j \in S$$

$$X(s,a) \geq 0 \quad \forall s \in S, \;\; a \in A_s.$$

Let $(X(s,a) : s \in S, a \in A_s)$ be a feasible solution for the dual (i.e., satisfies the constraints but not necessarily optimal). Every such feasible solution corresponds to a randomized Markov policy d^∞ and vice versa. Furthermore, for a given feasible solution, X, and the corresponding policy d^∞, $X(s,a)$ represents the expected total number of times you will be in state s and take action a following policy d^∞ before stopping in the indefinite-horizon problem. Thus, the objective in the dual can be interpreted as the total expected reward over the length of the indefinite horizon. The strong law of duality states that at the optimal solution, the objective functions in the primal and dual will be equal. But we already know that at the optimal, the primal objective will correspond to a weighted sum of $v^*_\lambda(s), s \in S$, which is the total expected discounted reward over the infinite (or indefinite) horizon given you start in state s. Thus our interpretations for the primal and dual variables coincide.

Solving the Infinite-Horizon Average-Reward MDP

Recall that in the average-reward model, the objective is to find the policy that has the maximum average reward, often called the *gain*. The gain of a policy can be written as

$$\begin{aligned} g^{\pi}(s) &= \lim_{n\to\infty} \frac{1}{N} v^{\pi}_{N+1} \\ &= \lim_{n\to\infty} \frac{1}{N} \sum_{n=1}^{N} [P_{\pi}^{n-1} r_{d_s}](s). \end{aligned} \quad (5)$$

As mentioned earlier, the major drawback is that for a given policy π, the gain may not even exist. An important result, however, states that if we confine ourselves to stationary policies, we can in fact be assured that the gain is well defined. Our ability to solve a given infinite- horizon average-reward problem depends on the form of the Markov chains induced by the deterministic, stationary policies available in the problem. Thus, we divide the set of average-reward MDPs according to the structure of the underlying Markov chains. We say that an MDP is

- *Unichain* if the transition matrix corresponding to *every* deterministic stationary policy is unichain, that is, it consists of a single recurrent class plus a possibly empty set of transient states, or
- *Multichain* if the transition matrix corresponding to *at least one* stationary policy contains two or more closed irreducible recurrent classes

If an MDP is unichain, then the gain for any given stationary, deterministic policy can be defined by a single number (independent of starting state). This makes intuitive sense since if we assume that it is possible to visit every state from every other one (possibly minus some set of transient states that may be visited initially but will eventually be abandoned), then it would seem reasonable to assume that over the infinite horizon, the initial starting state would not impact the average reward. However, if the initial state impacts what set of states can be visited in the future (i.e., the MDP is multichain), then clearly it is likely that the expected average reward will be dependent on the initial state.

If the average-reward MDP is unichain, then the *gain* can be uniquely determined by solving

$$v(s) = \max_{a\in A(s)} \left\{ r(s,a) - g + \sum_{s'\in S} p(s'|s,a) v(s') \right\}. \quad (6)$$

Notice that the above equation has $|S| + 1$ unknowns but only $|S|$ equations. Thus, v is not uniquely determined. To specify v uniquely, it is sufficient to set $v(s') = 0$ for some $s' \in S$. If this is done, then $v(s)$ is called the relative value function and $v(j) - v(k)$ is the difference in expected total reward obtained in using an optimal policy and starting in state j as opposed to state k. It is also often represented by the letter h and called the *bias*.

As in the discounted infinite-horizon MDP, there are three potential methods for solving the average-reward case. We present only policy iteration here and refer the reader to the recommended readings for value iteration and linear programming.

Policy Iteration

1. Set $n = 0$, and choose an arbitrary decision d_n.
2. (Policy evaluation) Solve for g_n, v_n:

$$0 = r_{d_n} - ge + (P_{d_n} - I)v.$$

3. Choose d_{n+1} to satisfy

$$d_{n+1} \in \text{argmax}_{d\in D}\{r_d + P_d v_n\}.$$

Setting $d_{n+1} = d_n$ if possible.

4. If $d_{n+1} = d_n$, stop, set $d^* = d_n$. Else, increment *n* by 1 and return to step 2.

As mentioned earlier, the equation in step 2 fails to provide a unique v_n since we have $|S| + 1$ unknowns and only $|S|$ equations. We therefore need an additional equation. Any one of the following three will suffice:

1. Set $v_n(s_0) = 0$ for some fixed $s_0 \in S$.
2. Choose v_n to satisfy $P^*_{d_n} v_n = 0$.
3. Choose v_n to satisfy $-v_n + (P_d - I)w = 0$ for some $w \in V$.

Continuous-Time Models

So far, we have assumed that decision epochs occur at regular intervals but clearly in many applications this is not the case. Consider, for instance, a queueing control model where the service rate can be adjusted in response to the size of the queue. It is reasonable to assume, however, that changing the service rate is only possible following the completion of a service. Thus, if the service time is random, then the decision epochs will occur at random time intervals. We will therefore turn our attention now to systems in which the state changes and decision epochs occur at random times. At the most general level, decisions can be made at any point in time, but we will focus on the subset of models for which decision epochs only occur at state transitions. It turns out that this is usually sufficient as the added benefit of being able to change decisions apart from state changes does not generally improve performance. Thus, the models we study generalize the discrete-time MDP models by:

1. Allowing, or requiring, the decision-maker to choose actions whenever the system changes state
2. Modeling the evolution of the system in continuous time, and
3. Allowing the time spent in a particular state to follow an arbitrary probability distribution

Semi-Markov decision processes (SMDP) are continuous-time models where decisions are made at some but not necessarily all state transitions. The most common subset of these, called exponential SMDPs, are SMDPs where the intertransition times are exponentially distributed.

We distinguish between two processes:

1. The natural process that monitors the state of the system as if it were observed continually through time and
2. The embedded Markov chain that monitors the evolution of the system at the decision epochs only

For instance, in a queueing control model, one may decide only to change the rate of service every time there is an arrival. Then the embedded Markov chain would only keep track of the system at each arrival while the natural process would keep track of all state changes – including both arrivals and departures.

While the actions are generally only going to depend on the state of the system at each decision epoch, it is possible that the rewards/costs to the system may depend on the natural process. Certainly, in the queueing control model, the cost to the system would go down as soon as a departure occurs. In discrete models, it was sufficient to let the reward depend on the current state s and the current action a and possibly the next state s'. However, in an SMDP, the natural process may change between now and the next decision epoch, and moreover, the time the process stays in a given state is no longer fixed. Thus we need to consider two types of rewards/costs. First, a lump-sum reward, $k(s, a)$, for taking action a when in state s. Second, a reward *rate*, $c(j, s, a)$, paid out for each time unit that the natural process spends in state j until the next decision epoch when the state at the last decision epoch was s and the action taken was a. Note that if we insist that every state transition triggers a decision epoch, we can reduce this to $c(s, a)$ since the system remains in s until the next decision epoch.

Before we can state our objective, we need to determine what we mean by discounting. Again, because we are dealing with continuous time so that decision epochs are not evenly spaced, it is not sufficient to have a fixed discount factor λ. Instead, we will discount future rewards at rate $e^{-\alpha t}$, for some $\alpha > 0$. If we let $\lambda = e^{-\alpha}$ (the discount rate for one time unit) then $\alpha = 0.11$ corresponds to $\lambda = 0.9$. Thus an α around 0.1 is commonly used.

We can now state our objective. We look to find a policy that maximizes the total expected discounted reward over the infinite horizon. There is an average-reward model for continuous-time models as well but we will not discuss that here. Given a policy π, we can write its total expected discounted reward as

$$v_\alpha^\pi(s) = E_s^\pi\left[\sum_{n=0}^{\infty} \mathrm{e}^{-\alpha\sigma_n}\left(K(X_n, Y_n) + \int_{\sigma_n}^{\sigma_{n+1}} \mathrm{e}^{-\alpha(t-\sigma_n)} c(W_t, X_n, Y_n)\, \mathrm{d}t\right)\right], \tag{7}$$

where X_n and Y_n are the random variables that represent the state and action at time n, respectively, W_t is the random variable that represents the state of the natural process at time t, and σ_n is the random time of the nth decision epoch. Again, if we assume that each state transition triggers a decision epoch, $X_n = W_t$ for all $t \in [\sigma_n, \sigma_{n+1})$. We seek to find a policy π such that

$$v_\alpha^\pi(s) = v_\alpha^*(s) = \max_{\pi \in \Pi^{HR}} v_\alpha^\pi(s) \tag{8}$$

for all $s \in S$. Perhaps surprisingly, (7) can be reduced to one that has the same form as in the discrete-time case for any SMDP. As a consequence, all the theory and the algorithms that worked in the discrete version can be transferred to the continuous model! Again, we refer the reader to the recommended readings for the details.

Extensions

Partially Observed MDPs

In some instances, the state of the system may not be directly observable, but instead, the decision-maker receives a signal from the system that provides information about the state. For example, in medical decision-making, the health-care provider will not know the patient's true health status but will have on hand some diagnostic information that may be related to the patient's true health. These problems are modeled from a Bayesian perspective. The decision-maker uses the signal to update his estimate of the probability distribution of the system state. He then bases his decision on this probability distribution. The computational methods for solving partially observed MDPs are significantly more complex than in the fully observable case and only small problems have been solved numerically.

Parameter-Adaptive Dynamic Programming

Often the transition probabilities in an MDP are derived from a system model, which is determined by a few parameters. Examples include demand distributions in inventory control and arrival and/or service distributions in queueing systems. In these cases, the forms of the distributions are known (e.g., Poisson for demand models and exponential for arrival or service models) but their parameter values are not. Herein, the decision-maker seeks a policy that combines *learning* with *control*. A Bayesian approach is used. The parameter is related to the system state through a likelihood function, and after observing the system state, the probability distribution on the parameter is updated. This updated probability distribution provides the basis for choosing a policy.

Approximate Dynamic Programming

Arguably the greatest challenge to implementing MDP theory in practice is "the curse of dimensionality." As the complexity of a problem grows, the amount of information that needs to be stored in the state space quickly reaches a point where the MDP is no longer computationally tractable. There now exist several methods for dealing with this problem, all of which are grouped under the title of approximate dynamic programming or neuro-dynamic programming. These potential methods begin by restricting the value function to a certain class of functions and then seeking to find the optimal value function within this class. A typical approximation scheme is based on the linear architecture:

$$v^*(s) \approx \tilde{v}(s,r) = \sum_{i=1}^{k} r_i \phi_i(s),$$

where $\phi_i(s), i = 1, \ldots, k$ are predefined basis functions that attempt to characterize the state space and r is a set of weights applied to the basis functions. This reduces the problem from one with $|S|$-dimensions to one with $|k|$-dimensions. The questions are (1) how do you determine what class of functions (determined by ϕ) to choose and (2) how to find the best approximate value function within the chosen class (i.e., the best values for r). The first question is still very much wide open.

Answers to the second question fall into two main camps. On the one hand, there are a number of methods that seek to iteratively improve the approximation through the simulation of sample paths of the decision process. The second method uses linear programming but restricts the value function to the approximate form. This reduces the number of variables in the primal to a reasonable number (equal to the number of basis functions chosen). One can then determine the optimal set of weights, r, through column generation. One of the major challenges facing approximate dynamic programming is that it is difficult to determine how close the approximate value function is to its true value. In other words, how much more reward might have been accumulated had the original MDP been solved directly? Though there are some attempts in the literature to answer this question, it remains a significant challenge.

Cross-References

▶ Markov Decision Processes
▶ Partially Observable Markov Decision Processes

Recommended Reading

Bertsekas D (2000) Dynamic programming and optimal control. Athena Scientific, Belmont

Bertsekas D, Tsitsiklis J (1996) Neuro-dynamic programming. Athena Scientific, Belmont

Feinberg E, Shwartz A (2002) Handbook of Markov decision processes. Kluwer Academic, Boston

Puterman M (1994) Markov decision processes. Wiley, New York

Sutton R, Barto A (1998) Reinforcement learning. MIT, Cambridge

Dynamic Programming for Relational Domains

▶ Symbolic Dynamic Programming

Dynamic Selection of Bias

▶ Metalearning

Dynamic Systems

The dynamic systems approach emphasizes the human, and animal, interaction with the environment. Interactions are described by partial differential equations. Attractors and limit cycles represent stable states which may be analogous to attribute-values.

E

EBL

► Explanation-Based Learning

Echo State Network

► Reservoir Computing

ECOC

► Error Correcting Output Codes

Edge Prediction

► Link Prediction

Efficient Exploration in Reinforcement Learning

John Langford
Microsoft Research, New York, NY, USA

Synonyms

PAC-MDP learning

Definition

An agent acting in a world makes observations, takes actions, and receives rewards for the actions taken. Given a history of such interactions, the agent must make the next choice of action so as to maximize the long-term sum of rewards. To do this well, an agent may take suboptimal actions which allow it to gather the information necessary to later take optimal or near-optimal actions with respect to maximizing the long-term sum of rewards. These information gathering actions are generally considered exploration actions.

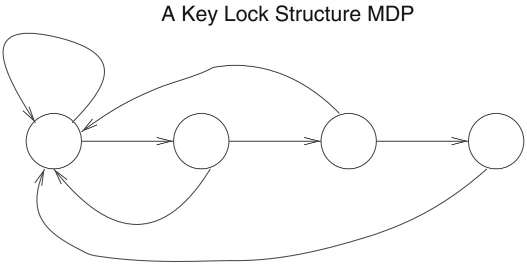
Motivation

Since gathering information about the world generally involves taking suboptimal actions compared with a later learned policy, minimizing the number of information gathering actions helps optimize the standard goal in reinforcement learning. In addition, understanding exploration well is key to understanding reinforcement learning well, since exploration is a key aspect of reinforcement learning which is missing from standard supervised learning settings (Fig. 1).

Efficient Exploration in Markov Decision Processes

One simplification of reinforcement learning is the ► Markov decision process setting. In this

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1



Efficient Exploration in Reinforcement Learning, Fig. 1 An example of a keylock MDP. The state are arranged in a chain. In each state, one of the two actions leads to the next state while the other leads back to the beginning. The only reward is in the transition to the last state in the chain. Keylock MDPs defeat simple greedy strategies, because the probability of randomly reaching the last transition is exponentially small in the length of the chain

setting, an agent repeatedly takes an action a, resulting in a transition to a state according to a conditional probability transition matrix $P(s'|s,a)$, and a (possibly probabilistic) reward $R(s',a,s) \in [0,1]$. The goal is to efficiently output a policy π which is ε-optimal over T timesteps. The value of policy π in a start state s is defined as

$$\eta(\pi, s) = E_{(a,s,r)^T \sim (\pi,P,R)^T} \sum_{t=1}^{T} r_t,$$

which should be read as the expectation over T-length sequences drawn from the interaction of the policy π with the world as represented by P and R. An ε-optimal policy π therefore satisfies:

$$\max_{\pi'} \eta(\pi', s) - \eta(\pi, s) \leq \varepsilon.$$

There are several notable results in this setting, typically expressed in terms of the dependence on the number of actions A, and the number of states S. The first is for the β-greedy strategy commonly applied when using ▶ Q-learning (Watkins and Dayan 1992) which explores randomly with probability β.

Theorem 1 *There exists MDPs such that with probability at least* $1/2$, *β-greedy requires* $\Theta(A^S)$ *explorations to find an ε-optimal policy.*

This is essentially a negative result, saying that a greedy exploration strategy cannot quickly discover a good policy in some settings. The proof uses an MDP with a key-lock like structure where for each state all actions but one take the agent back to the beginning state, and the reward is at the end of a chain of states.

It turns out that there exists algorithms capable of finding a near-optimal policy in an MDP with only a polynomial number of exploratory transitions.

Theorem 2 *For all MDPs, for any* $\delta > 0$, with probability $1-\delta$, the algorithm Explicit-Explore-or-Exploit finds an *ε-optimal policy after* $\tilde{O}(S^2A)$ *explorations.*

In other words, E^3 (Kearns and Singh 1998) requires exploration steps at most proportional to the size of the probability table driving the dynamics of the agent's world. The algorithm works in precisely the manner which might be expected: it builds a model of the world based on its observations and solves the model to determine whether to explore or exploit. The basic approach was generalized to stochastic games and reformulated as an "optimistic initialization" style algorithm named R-MAX (Brafman and Tennenholtz 2002).

It turns out that an even better dependence is possible using the delayed Q-learning (Strehl et al. 2006) algorithm.

Theorem 3 *For all MDPs, for any* $\delta > 0$, with probability $1-\delta$, the algorithm delayed Q-learning finds an *ε-optimal policy after* $\tilde{O}(SA)$ *explorations.*

The delayed Q-learning algorithm requires explorations proportional to the size of the solution policy rather than proportional to the size of world dynamics. At a high level, delayed Q-learning operates by keeping values for exploration and exploitation of observed state-actions, uses these values to decide between exploration and exploitation, and carefully updates these values. Delayed Q-learning does not obsolete E^3, because the (nonvisible) dependence on ε and T are worse (Strehl 2007).

This is a best possible result in terms of the dependence on S and A (up to log factors), as the following theorem (Kakade 2003) states:

Theorem 4 *For all algorithms, there exists an MDP such that with $\Omega(SA)$ explorations are required to find an ε optimal policy with probability at least $\frac{1}{2}$.*

Since even representing a policy requires a lookup table of size SA, this algorithm-independent lower bound is relatively unsurprising.

Variations on MDP Learning

There are several minor variations in the setting and goal definitions which do not qualitatively impact the set of provable results. For example, if rewards are in a bounded range, they can be offset and rescaled to the interval [0, 1].

It's also common to use a soft horizon (or discounting) where the policy evaluation is changed to:

$$\eta_\gamma(\pi, s) = E_{(a,s,r)^\infty \sim (\pi,P,R)^\infty} \sum_{t=1}^{\infty} \gamma^t r_t$$

for some value $\gamma < 1$. This setting is not precisely equivalent to the hard horizon, but since

$$sum^{\infty}_{t=(1n(1/\epsilon)+1n(1/1-\gamma))/1-\gamma} \gamma^t r_t \leq \varepsilon$$

similar results are provable with $1/(1-\gamma)$ taking the role of T and slightly altered algorithms.

One last variation changes the goal. Instead of outputting an ε-optimal policy for the next T timesteps, we could have an algorithm to handle both the exploration and exploitation, then retrospectively go back over a trace of experience and mark a subset of the actions as "exploration actions," with a guarantee that the remainder of the actions are according to an ε-optimal policy (Kakade 2003). Again, minor alterations to known algorithms in the above setting appear to work here.

Alternative Settings

There are several known analyzed variants of the basic setting formed by making additional assumptions about the world. This includes Factored MDPs (Kearns and Koller 1999), Metric MDPs (Kakade et al. 2003), Continuous MDPs (Brunskill et al. 2008), MDPs with a Bayesian prior (Poupart et al. 2006), and apprenticeship learning where there is access to a teacher for an MDP (Abbeel and Ng 2005). The structure of these results are all similar at a high level: with some additional information, it is possible to greatly ease the difficulty of exploration allowing tractable application to much larger problems.

Cross-References

▶ *k*-Armed Bandit
▶ Reinforcement Learning

Recommended Reading

Abbeel P, Ng A (2005) Exploration and apprenticeship learning in reinforcement learning. In: ICML 2005, Bonn

Brafman RI, Tennenholtz M (2002) R-MAX – a general polynomial time algorithm for near-optimal reinforcement learning. J Mach Learn Res 3:213–231

Brunskill E, Leffler BR, Li L, Littman ML, Roy N (2008) CORL: a continuous-state offset-dynamics reinforcement learner. In: UAI-08, Helsinki July 2008

Kakade S (2003) Thesis at gatsby computational neuroscience unit

Kakade S, Kearns M, Langford J (2003) Exploration in metric state spaces. In: ICML 2003, Washington, DC

Kearns M, Koller D (1999) Efficient reinforcement learning in factored MDPs. In: Proceedings of the 16th international joint conference on artificial intelligence. Morgan Kaufmann, San Francisco, pp 740–747

Kearns M, Singh S (1998) Near-optimal reinforcement learning in polynomial time. In: ICML 1998. Morgan Kaufmann, San Francisco, pp 260–268

Poupart P, Vlassis N, Hoey J, Regan K (2006) An analytic solution to discrete Bayesian reinforcement

learning. In: ICML 2006. ACM Press, New York, pp 697–704
Strehl A (2007) Thesis at Rutgers University
Strehl AL, Li L, Wiewiora E, Langford J, Littman ML (2006) PAC model-free reinforcement learning. In: Proceedings of the 23rd international conference on machine learning (ICML 2006), Pittsburgh, pp 881–888
Watkins C, Dayan P (1992) Q-learning. Mach Learn J 8:279–292

EFSC

▸ Evolutionary Feature Selection and Construction

Eigenvector

▸ K-Way Spectral Clustering

Elman Network

▸ Simple Recurrent Network

Embodied Evolutionary Learning

▸ Evolutionary Robotics

Emerging Patterns

Definition

Emerging pattern mining is an area of ▸ supervised descriptive rule induction. Emerging patterns are defined as itemsets whose support increases significantly from one data set to another (Dong 1999). Emerging patterns are said to capture emerging trends in time-stamped databases, or to capture differentiating characteristics between classes of data.

Recommended Reading

Dong G, Li J (1999) Efficient mining of emerging patterns: discovering trends and differences. In: Proceedings of the 5th ACM SIGKDD international conference on knowledge discovery and data mining (KDD-99), San Diego, pp 43–52

Empirical Risk Minimization

Xinhua Zhang
NICTA, Australian National University, Canberra, ACT, Australia
School of Computer Science, Australian National University, Canberra, ACT, Australia
NICTA London Circuit, Canberra, ACT, Australia

Definition

The goal of learning is usually to find a model which delivers good generalization performance over an underlying distribution of the data. Consider an input space $\mathcal{X}$ and output space $\mathcal{Y}$. Assume the pairs $(X \times Y) \in \mathcal{X} \times \mathcal{Y}$ are random variables whose (unknown) joint distribution is P_{XY}. It is our goal to find a predictor $f : \mathcal{X} \mapsto \mathcal{Y}$ which minimizes the expected risk:

$$P(f(X) \neq Y) = \mathsf{E}_{(X,Y)\sim P_{XY}} \left[\delta(f(X) \neq Y)\right],$$

where $\delta(z) = 1$ if z is true, and 0 otherwise.

However, in practice we only have n pairs of training examples (X_i, Y_i) drawn identically and independently from P_{XY}. Since P_{XY} is unknown, we often use the risk on the training set (called empirical risk) as a surrogate of the expected risk on the underlying distribution:

$$\frac{1}{n}\sum_{i=1}^{n} \delta(f(X_i) \neq Y_i).$$

Empirical Risk Minimization (ERM) refers to the idea of choosing a function f by minimizing the empirical risk. Although it is often effective and efficient, ERM is subject to ▶ overfitting, i.e. finding a model which fits the training data well but predicts poorly on unseen data. Therefore, ▶ regularization is often required.

More details about ERM can be found in Vapnik (1998).

Recommended Reading

Vapnik V (1998) Statistical learning theory. John Wiley and Sons, New York

Ensemble Learning

Gavin Brown
The University of Manchester, Manchester, UK

Synonyms

Committee machines; Multiple classifier systems

Definition

Ensemble learning refers to the procedures employed to train multiple learning machines and combine their outputs, treating them as a "committee" of decision makers. The principle is that the decision of the committee, with individual predictions combined appropriately, should have better overall ▶ accuracy, on average, than any individual committee member. Numerous empirical and theoretical studies have demonstrated that ensemble models very often attain higher accuracy than single models.

The members of the ensemble might be predicting real-valued numbers, class labels, posterior probabilities, rankings, clusterings, or any other quantity. Therefore, their decisions can be combined by many methods, including averaging, voting, and probabilistic methods. The majority of ensemble learning methods are generic, applicable across broad classes of model types and learning tasks.

Motivation and Background

If we could build the "perfect" machine learning device, one which would give us the best possible answer every time, there would be no need for *ensemble learning* methods – indeed, there would be no need for this encyclopedia either. The underlying principle of ensemble learning is a recognition that in real-world situations, every model has limitations and will make errors. Given that each model has these "limitations," the aim of ensemble learning is to manage their strengths and weaknesses, leading to the best possible decision being taken overall. Several theoretical and empirical results have shown that the accuracy of an ensemble can significantly exceed that of a single model.

The principle of combining predictions has been of interest to several fields over many years. Over 200 years ago, a controversial question had arisen, on how best to estimate the mean of a probability distribution given a small number of sample observations. Laplace (1818) demonstrated that the sample mean was not always optimal: under a simple condition, the sample median was a better combined predictor of the population mean. The financial forecasting community has analyzed model combination for several decades, in the context of stock portfolios. The contribution of the machine learning (ML) community emerged in the 1990s – automatic construction (from data) of both the models and the method to combine them. While the majority of the ML literature on this topic is from 1990 onward, the principle has been explored briefly by several independent authors since the 1960s. See Kuncheva (2004b) for historical accounts.

The study of ensemble methods, with model outputs considered for their abstract properties rather than the specifics of the algorithm which produced them, allows for a wide impact across many fields of study. If we can understand precisely why, when, and how particular ensemble methods can be applied successfully, we would have made progress toward a powerful new tool for Machine Learning: *the ability to automatically exploit the strengths and weaknesses of different learning systems.*

Methods and Algorithms

An ensemble consists of a set of models and a method to combine them. We begin this section by assuming that we have a set of models, generated by any of the learning algorithms in this encyclopedia; we explore popular methods of combining their outputs, for classification and regression problems. Following this, we review some of the most popular ensemble algorithms, for *learning* a set of models given the knowledge that they will be combined, including extensive pointers for further reading. Finally, we take a theoretical perspective, and review the concept of ensemble *diversity*, the fundamental property which governs how well an ensemble can perform.

Methods for Combining a Set of Models

There exist numerous methods for model combination, far too many to fully detail here. The *linear* combiner, the *product* combiner, and the *voting* combiner are by far the most commonly used in practice. Though a combiner could be specifically chosen to optimize performance in a particular application, these three rules have shown consistently good behavior across many problems, and are simple enough that they are amenable to theoretical analysis.

The linear combiner is used for models that output real-valued numbers, so is applicable for ▸ regression ensembles, or for ▸ classification ensembles producing class probability estimates. Here, notation for the latter case is only shown. We have a model $f_t(y|\mathbf{x})$, an estimate of the probability of class y given input $\mathbf{x}$. For a set of these, $t = \{1, \ldots, \mathrm{T}\}$, the ensemble probability estimate is,

$$\bar{f}(y|\mathbf{x}) = \sum_{t=1}^{T} w_t f_t(y|\mathbf{x}). \tag{1}$$

If the weights $w_t = 1/T, \forall t$, this is a simple uniform averaging of the probability estimates. The notation clearly allows for the possibility of a nonuniformly weighted average. If the classifiers have different accuracies on the data, a nonuniform combination could *in theory* give a lower error than a uniform combination. However, in practice, the difficulty of estimating the $\mathbf{w}$ parameters without overfitting, and the relatively small gain that is available (see Kuncheva 2004b, p. 282), have meant that in practice the uniformly weighted average is by far the most commonly used. A notable exception, to be discussed later in this article, is the *mixture of experts* paradigm – in MoE, weights are nonuniform, but are learnt and dependent on the input value $\mathbf{x}$. An alternative combiner is the *product rule*:

$$\bar{f}(y|\mathbf{x}) = \frac{1}{Z} \prod_{t=1}^{T} f_t(y|\mathbf{x})^{wt}, \tag{2}$$

where Z is a normalization factor to ensure $\bar{f}$ is a valid distribution. Note that Z is not *required* to make a valid decision, as the order of posterior estimates remain unchanged before/after normalization. Under the assumption that the class-conditional probability estimates are independent, this is the theoretically optimal combination strategy. However, this assumption is highly unlikely to hold in practice, and again the weights $\mathbf{w}$ are difficult to reliably determine. Interestingly, the linear and product combiners are in fact special cases of the *generalized mean* (Kuncheva 2004b) allowing for a continuum of possible combining strategies.

The linear and product combiners are applicable when our models output real-valued numbers. When the models instead output class labels, a majority (or plurality) vote can be used. Here, each classifier votes for a particular class, and the class with the most votes is chosen as the ensemble output. For a two-class problem the models produce labels, $h_t(\mathbf{x}) \in \{-1, +1\}$. In this case, the ensemble output for the *voting* combiner can be written as

$$H(\mathbf{x}) = sign\left(\sum_{t=1}^{T} w_t h_t(x)\right). \tag{3}$$

The weights $\mathbf{w}$ can be uniform for a simple majority vote, or nonuniform for a weighted vote.

We have discussed only a small fraction of the possible combiner rules. Numerous other rules exist, including methods for combining rankings of classes, and unsupervised methods to combine clustering results. For details of the wider literature, see Kuncheva (2004b) or Polikar (2006).

Algorithms for Learning a Set of Models

If we had a committee of *people* taking decisions, it is self-evident that we would not want them all to make the same bad judgments *at the same time*. With a committee of learning models, the same intuition applies: we will have no gain from combining a set of identical models. We wish the models to exhibit a certain element of "diversity" in their group behavior, though still retaining good performance individually.

We therefore make a distinction between two types of ensemble learning algorithms, those which encourage diversity *implicitly*, and those which encourage it *explicitly*. The vast majority of ensemble methods are *implicit*, in that they provide different *random subsets* of the training data to each learner. Diversity is encouraged "implicitly" by *random* sampling of the data space: at no point is a *measurement* taken to ensure diversity will emerge. The random differences between the datasets might be in the selection of examples (the ▶ Bagging algorithm), the selection of features (▶ Random Subspace Method, Ho (1998) or ▶ Rotation Forests, Rodriguez et al. 2006), or combinations of the two (the Random Forests algorithm, Breiman 2001). Many other "randomization" schemes are of course possible.

An alternative is to *explicitly* encourage diversity, constructing each ensemble member with some measurement ensuring that it is substantially different from the other members. ▶ Boosting algorithms achieve this by altering the distribution of training examples for each learner such that it is encouraged to make more accurate predictions where previous predictors have made errors. The DECORATE algorithm (Melville and Mooney 2005) explicitly alters the distribution of class labels, such that successive models are forced to learn different answers to the same problem. ▶ Negative correlation learning (see Brown 2004; Brown et al. 2005), includes a penalty term when learning each ensemble member, explicitly *managing* the accuracy-diversity trade-off.

In general, ensemble methods constitute a large class of algorithms – some based on heuristics, and some on sound learning-theoretic principles. The three algorithms that have received the most attention in the literature are reviewed here. It should be noted that we present only the most basic form of each; numerous modifications have been proposed for a variety of learning scenarios. As further study the reader is referred to the many comprehensive surveys of the field (Brown et al. 2005; Kuncheva 2004b; Polikar 2006).

Bagging

In the Bagging algorithm (Breiman 1996) each member of the ensemble is constructed from a different training dataset, and the predictions combined either by uniform averaging or voting over class labels. Each dataset is generated by sampling from the total N data examples, choosing N items uniformly at random *with replacement*. Each sample is known as a *bootstrap*; the name Bagging is an acronym derived from *B*ootstrap *AGG*regat*ING*. Since a bootstrap samples N items uniformly at random with replacement, the probability of any individual data item *not* being selected is $p = (1 - 1/N)^N$. Therefore with large N, a single bootstrap is expected to contain approximately 63. 2 % of the original set, while 36. 8 % of the originals are not selected.

Like many ensemble methods, Bagging works best with *unstable* models, that is those that produce differing generalization behavior with small changes to the training data. These are also known as *high variance* models, examples of which are ▶ decision trees and ▶ neural networks. Bagging therefore tends not to work well with very simple models. In effect, Bagging samples randomly from the space of possible models to make up the ensemble – with very simple models the sampling produces almost identical (low diversity) predictions.

Despite its apparent capability for variance reduction, situations have been demonstrated where

Algorithm 1 Bagging

```
Input: Required ensemble size $T$
Input: Training set $S = \{(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N)\}$
for $t = 1$ to $T$ do
    Build a dataset $S_t$, by sampling $N$ items, randomly with replacement from $S$.
    Train a model $h_t$ using $S_t$, and add it to the ensemble
end for
For a new testing point $(x', y')$,
If model outputs are continuous, combine them by voting.
```

Algorithm 2 Adaboost

```
Input: Required ensemble size $T$
Input: Training set $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, where $y_j \in \{-1, +1\}$
Define a uniform distribution $D_1(i)$ over elements of $S$.
for $t = 1$ to $T$ do
    Train a model $h_t$ using distribution $D_t$.
    Calculate $e_t = P_{D_t}(h_t(x) \neq y)$
    If $\epsilon_t \geq 0.5$ break
    Set $\alpha_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
    Update $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
where $Z_t$ is a normalization factor so that $D_{t+1}$ is a valid distribution.
end for
For a new testing point $(x', y')$ ,
$H(x') = \text{sign}(\Sigma_{t=1}^{T} \alpha_t h_t(x'))$
```

Bagging can converge *without* affecting variance (see Brown et al. 2005). Several other explanations have been proposed for Bagging's success, including links to Bayesian model averaging. In summary, it seems that several years from its introduction, despite its apparent simplicity, Bagging is still not fully understood.

Adaboost

Adaboost (Freund and Schapire 1996) is the most well known of the *Boosting* family of algorithms (Schapire 2003). The algorithm trains models sequentially, with a new model trained at each round. At the end of each round, mis-classified examples are identified and have their emphasis increased in a new training set which is then fed back into the start of the next round, and a new model is trained. The idea is that subsequent models should be able to compensate for errors made by earlier models.

Adaboost occupies somewhat of a special place in the history of ensemble methods. Though the procedure seems heuristic, the algorithm is in fact grounded in a rich learning-theoretic body of literature. (Schapire 1990) addressed a question posed by Kearns and Valiant (1988) on the nature of two complexity classes of learning problems. The two classes are *strongly learnable* and *weakly learnable* problems. Schapire showed that these classes were equivalent; this had the corollary that a weak model, performing only slightly better than random guessing, could be "boosted" into an arbitrarily accurate *strong* model. The original Boosting algorithm was a proof by construction of this equivalence, though had a number of impractical assumptions built-in. The Adaboost algorithm (Freund and Schapire 1996) was the first practical Boosting method. The authoritative historical account of the development can be found in Schapire (1999), including discussion of numerous variants and interpretations of the algorithm. The procedure is shown in Algorithm 2. Some similarities with Bagging are evident; a key differences is that at each round t, Bagging has a uniform distribution D_t, while Adaboost adapts a nonuniform distribution.

The ensemble is constructed by iteratively adding models. Each time a model is learnt, it is checked to ensure it has at least $\varepsilon_t < 0.5$, that is, it has performance *better than random guessing* on the data it was supplied with. If it does not, either an alternative model is constructed, or the loop is terminated.

After each round, the distribution D_t is updated to emphasize incorrectly classified examples. The update causes half the distribution mass of D_{t+1} to be over the examples incorrectly classified by the previous model. More precisely, $\sum_{h_t(x_1) \neq y_i} D_{t+1}(i) = 0.5$. Thus, if h_t has an error rate of 10 %, then examples from that small 10 % will be allocated 50 % of the next model's training "effort," while the remaining examples (those correctly classified) are underemphasized. An equivalent (and simpler) writing of the distribution update scheme is to multiply $D_t(i)$ by $1/2(1 - \varepsilon_t)$ if $h_t(x_i)$ is correct, and by $1/2\varepsilon_t$ otherwise.

The updates cause the models to sequentially minimize an exponential bound on the error rate. The training error rate on a data sample $\mathcal{S}$ drawn from the true distribution $\mathcal{D}$ obeys the bound,

$$P_{x,y\sim\mathcal{S}}(yH(x) < 0) \le \prod_{t=1}^{T} 2\sqrt{\varepsilon_t(1-\varepsilon_t)}. \quad (4)$$

This *upper bound* on the training error (though not the *actual* training error) is guaranteed to decrease monotonically with T, given $\varepsilon_t < 0.5$.

In an attempt to further explain the performance of Boosting algorithms, Schapire also developed bounds on the *generalization* error of voting systems, in terms of the voting margin, the definition of which was given in (10). Note that, this is not the same as the *geometric margin*, optimized by ▸ support vector machines. The difference is that the voting margin is defined using the one-norm $||\mathbf{w}||_1$ in the denominator, while the geometric margin uses the *two*-norm $||\mathbf{w}||_2$. While this is a subtle difference, it is an important one, forming links between SVMs and Boosting algorithms – see Rätsch et al. (2002) for details. The following bound holds with probability $1-\delta$,

$$Px, y \sim \mathcal{D}(H(x) \neq y)$$
$$\le P_{x,y\sim\mathcal{S}}(yH(x) < \theta) + \tilde{O}\left(\sqrt{\frac{d}{N\theta^2} - ln\delta}\right), \quad (5)$$

where the $\tilde{O}$ notation hides constants and logarithmic terms, and d is the ▸ VC-dimension of the model used. Roughly, this states that the generalization error is less than or equal to the training error plus a term dependent on the voting margin. The larger the minimum margin in the training data, the lower the testing error. The original bounds have since been significantly improved, see Koltchinskii and Panchenko (2005) as a comprehensive recent work. We note that this bound holds generally for *any* voting system, and is not specific to the Boosting framework.

The margin-based theory is only one explanation of the success of Boosting algorithms. Mease and Wyner (2008) present a discussion of several questions on why and how Adaboost succeeds. The subsequent 70 pages of discussion demonstrate that the story is by no means simple. The conclusion is, while no single theory can fully explain Boosting, each provides a different part of the still unfolding story.

Mixtures of Experts

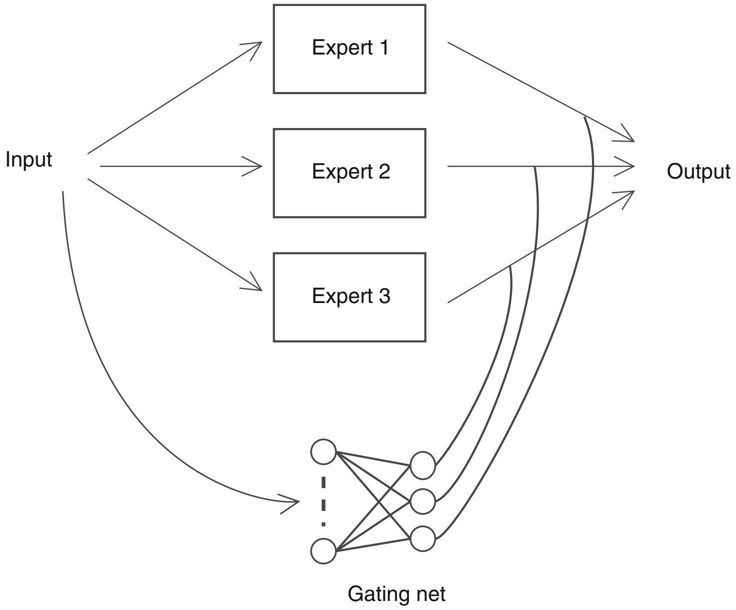
The mixtures of experts architecture is a widely investigated paradigm for creating a combination of models (Jacobs et al. 1991). The principle underlying the architecture is that certain models will be able to "specialize" to particular parts of the input space. It is commonly implemented with a neural network as the base model, or some other model capable of estimating probabilities. A *Gating network* receives the same inputs as the component models, but its outputs are used as the weights for a linear combiner. The Gating network is responsible for learning the appropriate weighted combination of the specialized models ("experts") for any given input. Thus, the input space is "carved-up" between the experts, increasing and decreasing their weights for particular examples. In effect, a mixture of experts explicitly learns how to create expert ensemble members in different portions of the input space, and select the most appropriate subset for a new testing example (Fig. 1).

The architecture has received wide attention, and has a strong following in the probabilistic modeling community, where it may go under the pseudonym of a "mixture model." A common training method is the ▸ expectation-maximization algorithm.

Theoretical Perspectives: Ensemble Diversity

We have seen that all ensemble algorithms in some way attempt to encourage "diversity." In

Ensemble Learning, Fig. 1 The mixtures of experts architecture



this section, we take a more formalized perspective, to understand what is meant by this term.

What Is Diversity?

The optimal "diversity" is fundamentally a *credit assignment* problem. If the committee as a whole makes an erroneous prediction, how much of this error should be attributed to each member? More precisely, how much of the committee prediction is due to the accuracies of the individual models, and how much is due to their interactions when they were combined? We would ideally like to reexpress the ensemble error as two distinct components: a term for the accuracies of the individual models, plus a term for their interactions, i.e., their *diversity*.

It turns out that this so-called *accuracy-diversity* breakdown of the ensemble error is not always possible, depending on the type of error function, and choice of combiner rule. It should be noted that when "diversity" is referred to in the literature, it is most often meant to indicate classification with a majority vote combiner, but for completeness we address the general case here. In the following sections, the existing work to understand diversity in three distinct cases is described: for regression tasks (a linear combiner), and classification tasks, with either a linear combiner or a voting combiner.

Regression Error with a Linear Combination Rule

In a regression problem, it is common to use the squared error criterion. The accuracy-diversity breakdown for this case (using a linear combiner) is called the *ambiguity decomposition* (Krogh and Vedelsby 1995). The result states that the squared error of the linearly combined ensemble, $\bar{f}(\mathbf{x})$, can be broken into a sum of two components:

$$(\bar{f}(\mathbf{x}) - d)^2 = \frac{1}{T}\sum_{t=1}^{T}(f_t(\mathbf{x}) - d)^2 - \frac{1}{T}\sum_{t=1}^{T}(f_t(\mathbf{x}) - \bar{f}(\mathbf{x}))^2. \quad (6)$$

The first term on the right hand side is the average squared error of the individual models, while the second term quantifies the interactions *between* the predictions. Note that this second term, the "ambiguity," is always positive. This guarantees that, for an arbitrary data point, the ensemble squared error is always less than or equal to the average of the individual squared errors.

The intuition here can be understood as follows. Imagine five friends, playing "guess the weight of the cake" (an old English fairground game): if a player's guess is close enough to the true weight, they win the cake. Just as they are about to play, the fairground manager states that they can only submit *one* guess. The dilemma seems to be in whose guess they should submit – however, the ambiguity decomposition shows us that taking the average of their guesses, and submitting that, will *always* be closer (on average) than choosing a person at random and submitting their guess. Note that this is qualified with "on average" – it may well be that one of the predictions will in fact be closer than the average prediction, but we presume that we have no way of identifying *which* prediction to choose, other than random. It can be seen that greater diversity in the predictions (i.e., a larger ambiguity term) results in a larger gain over the average individual performance. However, it is also clear that there is a trade-off to be had: too much diversity and the average error is extremely large.

The idea of a trade-off between these two terms is reminiscent of the ▸ bias-variance decomposition (Geman et al. 1992); in fact, there is a deep connection between these results. Taking the expected value of (6) over all possible training sets gives us the ensemble analogy to the bias-variance decomposition, called the ▸ bias-variance-covariance decomposition (Ueda and Nakano 1996). This shows that the expected squared error of an ensemble $\bar{f}(\mathbf{x})$ from a target d is:

$$\mathcal{E}_{\mathcal{D}}\{(\bar{f}(\mathbf{x})-d)^2\} = \overline{\text{bias}}^2 + \frac{1}{T}\overline{\text{var}} + \left(1-\frac{1}{T}\right)\overline{\text{covar}}, \qquad (7)$$

where the expectation is with respect to all possible training datasets $\mathcal{D}$. While the bias and variance terms are constrained to be positive, the covariance between models can become negative – thus the definition of diversity emerges as an extra degree of freedom in the bias-variance dilemma. This extra degree of freedom allows an ensemble to approximate functions that are difficult (if not impossible) to find with a single model. See Brown et al. (2005) for extensive further discussion of this concept.

Classification Error with a Linear Combination Rule

In a classification problem, our error criterion is the misclassification rate, also known as the *zero-one* loss function. For this type of loss, it is well known there is no unique definition of bias-variance; instead there exist multiple decompositions each with advantages and disadvantages (see Kuncheva 2004b, p. 224). This gives us a clue as to the situation with an ensemble – there is also no simple accuracy-diversity separation of the ensemble classification error. Classification problems can of course be addressed either by a model producing class probabilities (where we linearly combine), or directly producing class labels (where we use majority vote). Partial theory has been developed for each case.

For linear combiners, there exist theoretical results that relate the correlation of the probability estimates to the ensemble classification error. Tumer and Ghosh (1996) showed that the reducible classification error (i.e., above the Bayes rate) of a simple averaging ensemble, e_{ave}, can be written as

$$e_{\text{ave}} = e_{\text{add}}\left(\frac{1+\delta(T-1)}{T}\right), \qquad (8)$$

where e_{add} is the classification error of an individual model. The δ is a correlation coefficient between the model outputs. When the individual models are identical, the correlation is $\delta = 1$. In this case, the ensemble error is equal to the individual error, $e_{\text{ave}} = e_{\text{add}}$. When the models are statistically independent, $\delta = 0$, and the ensemble error is a fraction $1/T$ of the individual error, $e_{\text{ave}} = 1/T \times \text{e}_{\text{add}}$. When δ is negative, the models are negatively correlated, and the ensemble error is lower than the average individual error. However, (8) is derived under quite strict assumptions, holding only for a local area around the decision boundary, and ultimately resting on the bias-variance-covariance theory from regression problems. Further details, including recent work to lift some of the assumptions (Kuncheva 2004b).

Classification Error with a Voting Combination Rule

The case of a classification problem with a majority vote combiner is the most challenging of all. In general, there is no known breakdown of the ensemble classification error into neat accuracy and diversity components. The simplest intuition to show that correlation between models does affect performance is given by the Binomial theorem. If we have T models each with identical error probability $p = P(h_t(\mathbf{x}) \neq y)$, assuming they make statistically *independent* errors, the following error probability of the majority voting committee holds,

$$P(H(x) \neq y) = \sum_{k>T/2}^{T} \binom{T}{k} p^k (1-p)^{T-k}. \tag{9}$$

For example, in the case of $T = 21$ ensemble members, each with error $p = 0.3$, the majority voting error will be $0.\,026$, an order of magnitude improvement over the individual error. However, this *only* holds for statistically independent errors. The correlated case is an open problem. Instead, various authors have proposed their own heuristic definitions of diversity in majority voting ensembles. Kuncheva (2004b) conducted extensive studies of several suggested diversity measures; the conclusion was that "*no measure consistently correlates well with the majority vote accuracy.*" In spite of this, some were found useful as an approximate guide to characterize performance of ensemble methods, though should not be relied upon as the "final word" on diversity. Kuncheva's recommendation in this case is the *Q-statistic* (Kuncheva 2004b, p. 299), due to its simplicity and ease of computation.

Breiman (2001) took an alternative approach, deriving not a *separation* of error components, but a *bound* on the generalization error of a voting ensemble, expressed in terms of the correlations of the models. To understand this, we must introduce concept of *voting margin*. The voting margin for a two-class problem, with $y \in \{-1, +1\}$, is defined,

$$m = \frac{y_t \sum_{t=1}^{T} w_t h_t(\mathbf{x})}{\sum_{t=1}^{T} |w_t|} = yH(\mathbf{x}). \tag{10}$$

If the margin is positive, the example is correctly classified, if it is negative, the example is incorrectly classified. The expected margin $s = \mathcal{E}_{\mathcal{D}}\{m\}$ measures the extent to which the average number of votes for the correct class exceeds the average vote for any other class, with respect to the data distribution $\mathcal{D}$. The larger the voting margin, the more confidence in the classification. Breiman's bound shows,

$$P_{\mathcal{D}}(H(\mathbf{x}) \neq y) = P_{\mathcal{D}}(yH(\mathbf{x}) < 0) \neq \frac{\bar{\rho}(1-s^2)}{s^2}. \tag{11}$$

Here $\bar{\rho}$ is the average pairwise correlation between the errors of the individual models. Thus, the generalization error is minimized by a small $\bar{\rho}$, and an s as close to 1 as possible. The balance between a high accuracy (large s) and a high diversity (low $\bar{\rho}$) constitutes the tradeoff in this case, although the bound is quite loose.

Summary

In summary, the definition of diversity depends on the problem. In a regression problem, the optimal diversity is the trade-off between the bias, variance and covariance components of the squared error. In a classification problem, with a linear combiner, there exists partial theory to relate the classifier correlations to the ensemble error rate. In a classification problem with a voting combiner, there is no single theoretical framework or definition of diversity. However, the lack of an agreed definition of diversity has not discouraged researchers from trying to achieve it, nor has it stalled the progress of effective algorithms in the field.

Conclusions and Current Directions in the Field

Ensemble methods constitute some of the most robust and accurate learning algorithms of the past decade (Caruana and Niculescu-Mizil 2006). A multitude of heuristics have been developed for randomizing the ensemble parameters, to generate diverse models. It is arguable that this line of investigation is nowadays rather oversubscribed, and the more interesting research is now in methods for nonstandard data. ▶ Cluster ensembles (Strehl and Ghosh 2003) are ensemble techniques applied to unsupervised learning problems. Problems with *nonstationary* data, also known as *concept drift*, are receiving much recent attention (Kuncheva 2004a). The most up to date innovations are to be found in the biennial *International Workshop on Multiple Classifier Systems* (Roli et al. 2000).

Recommended Reading

Kuncheva (2004b) is the standard reference in the field, which includes references to many further recommended readings. In addition, Brown et al. (2005) and Polikar (2006) provide extensive literature surveys. Roli et al. (2000) is an international workshop series dedicated to ensemble learning.

Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Brown G (2004) Diversity in neural network ensembles. PhD thesis, University of Birmingham

Brown G, Wyatt JL, Harris R, Yao X (2005) Diversity creation methods: a survey and categorisation. J Inf Fusion 6(1):5–20

Caruana R, Niculescu-Mizil A (2006) An empirical comparison of supervised learning algorithms. In: Proceedings of the 23rd international conference on machine learning. ACM, New York, pp 161–168

Freund Y, Schapire R (1996) Experiments with a new boosting algorithm. In: Proceedings of the thirteenth international conference on machine learning (ICML'96). Morgan Kauffman Publishers, San Francisco, pp 148–156

Geman S, Bienenstock E, Doursat R (1992) Neural networks and the bias/variance dilemma. Neural Comput 4(1):1–58

Ho TK (1998) The random subspace method for constructing decision forests. IEEE Trans Pattern Anal Mach Intell 20(8):832–844

Jacobs RA, Jordan MI, Nowlan SJ, Hinton GE (1991) Adaptive mixtures of local experts. Neural Comput 3(1):79–87

Kearns M, Valiant LG (1988) Learning Boolean formulae or finite automata is as hard as factoring. Technical report TR-14-88, Harvard University Aiken Computation Laboratory

Koltchinskii V, Panchenko D (2005) Complexities of convex combinations and bounding the generalization error in classification. Ann Stat 33(4):1455

Krogh A, Vedelsby J (1995) Neural network ensembles, crossvalidation and active learning. In: Advances in neural information processing systems. MIT Press, Cambridge, pp 231–238

Kuncheva LI (2004a) Classifier ensembles for changing environments. In: International workshop on multiple classifier systems. Lecture notes in computer science, vol 3007. Springer, Berlin

Kuncheva LI (2004b) Combining pattern classifiers: methods and algorithms. Wiley, New York

Laplace PS (1818) Deuxieme supplement a la theorie analytique des probabilites. Gauthier-Villars, Paris

Mease D, Wyner A (2008) Evidence contrary to the statistical view of Boosting. J Mach Learn Res 9:131–156

Melville P, Mooney RJ (2005) Creating diversity in ensembles using artificial data. Inf Fusion 6(1):99–111

Polikar R (2006) Ensemble based systems in decision making. IEEE Circ Syst Mag 6(3):21–45

Rätsch G, Mika S, Schölkopf B, Müller KR (2002) Constructing Boosting algorithms from SVMs: an application to one-class classification. IEEE Trans Pattern Anal Mach Intell 24(9):1184–1199

Rodriguez J, Kuncheva L, Alonso C (2006) Rotation forest: a new classifier ensemble method. IEEE Trans Pattern Anal Mach Intell 28(10):1619–1630

Roli F, Kittler J, Windridge D, Oza N, Polikar R, Haindl M et al (eds) Proceedings of the international workshop on multiple classifier systems 2000–2009. Lecture notes in computer science. Springer, Berlin. Available at: http://www.informatik.uni-trier.de/ley/db/conf/mcs/index.html

Schapire RE (1990) The strength of weak learnability. Mach Learn 5:197–227

Schapire RE (1999) A brief introduction to boosting. In: Proceedings of the 16th international joint conference on artificial intelligence. Morgan Kaufmann, San Francisco, pp 1401–1406

Schapire RE (2003) The boosting approach to machine learning: an overview. In: Denison DD, Hansen MH, Holmes C, Mallick B, Yu B (eds) Nonlinear estimation & classification Lecture notes in statistics. Springer, Berlin, pp 149–172

Strehl A, Ghosh J (2003) Cluster ensembles – a knowledge reuse framework for combining multiple partitions. J Mach Learn Res 3:583–617

Tumer K, Ghosh J (1996) Error correlation and error reduction in ensemble classifiers. Connect Sci 8(3–4):385–403

Ueda N, Nakano R (1996) Generalization error of ensemble estimators. In: Proceedings of IEEE international conference on neural networks, vol 1, pp 90–95. ISBN:0-7803-3210-5

Entailment

Synonyms

Implication; Logical consequence

Definition

The term entailment is used in the context of logical reasoning. Formally, a logical formula T entails a formula c if and only if all models of T are also a model of c. This is usually denoted as $T \models c$ and means that c is a logical consequence of T or that c is implied by T.

Let us elaborate this definition for propositional clausal logic, where the formulae T could be the following expression:

```
flies :- bird, normal.
bird :- blackbird.
bird :- ostrich.
```

Here, the first clause or rule can be read as flies *if* normal *and* bird, that is, normal birds fly, the second and third one as stating that blackbirds, resp. ostriches, are birds. An interpretation is then an assignment of truth-values to the propositional variables. For instance, for the above domain

```
{ostrich, bird}
{blackbird, bird, normal}
```

are interpretations, specified through the set of propositional variables that are true. This means that in the first interpretation, the only true propositions are `ostrich` and `bird`. An interpretation specifies a kind of possible world. An interpretation I is then a model for a clause $h : -b_1, \dots b_n$ if and only if $\{b_1, \dots, b_n\} \subseteq I \rightarrow h \in I$ and it is model for a clausal theory if and only if it is a model for all clauses in the theory. Therefore, the first interpretation above is a model for the theory, but the second one is not because the interpretation is not a model for the first clause (as $\{$`bird, normal`$\} \subseteq I$ but `flies` $\notin I$). Using these notions, it can now be verified that the clausal theory T above logically entails the clause

```
flies :- ostrich, normal.
```

because all models of the theory are also a model for this clause.

In machine learning, the notion of entailment is used as a covers relation in ▸ inductive logic programming, where hypotheses are clausal theories, instances are clauses, and an example is covered by the hypothesis when it is entailed by the hypothesis.

Cross-References

▸ Inverse Entailment
▸ Logic of Generality

Recommended Reading

Russell S, Norvig P (1995) Artificial intelligence: a modern approach, 2nd edn. Prentice Hall, Englewood Cliffs

Entity Resolution

Indrajit Bhattacharya[1] and Lise Getoor[2]
[1]IBM India Research Laboratory, New Delhi, India
[2]University of Maryland, College Park, MD, USA

Abstract

References to real-world entities are often ambiguous, more commonly across data sources but frequently within a single data source as well. Ambiguities occur due to multiple reasons, such as incorrect data entry, or multiple possible representations of the entities. Given such a collection of ambiguous entity references, the goal of entity resolution

is to discover the unique set of underlying entities, and map each reference to its corresponding entity. Resolving such entity ambiguities is necessary for removing redundancy and also for accurate entity-level analysis. This is a common problem that comes up in many different applications and has been studied in different branches of computer science. As evidences for entity resolution, traditional approaches consider pair-wise similarity between references, and many sophisticated similarity measures have been proposed to compare attributes of references. The simplest solution classifies reference pairs with similarity above a threshold as referring to the same entity. More sophisticated solutions use a probabilistic framework for reasoning with the pair-wise probabilities. Recently proposed relational approaches for entity resolution make use of relationships between references when available as additional evidences. Instead of reasoning independently for each pair of references, these approaches reason collectively over related pair-wise decisions over references. One line of work within the relational family uses supervised or unsupervised probabilistic learning using probabilistic graphical models, while another uses more scalable greedy techniques for merging references in a hyper-graph. Beyond improving entity resolution accuracy, such relational approaches yield additional knowledge in the form of relationships between the underlying entities.

Synonyms

Co-reference resolution; Deduplication; Duplicate detection; Identity uncertainty; Merge-purge; Object consolidation; Record linkage; Reference reconciliation

Definition

A fundamental problem in data cleaning and integration (see ▶ Data Preparation) is dealing with uncertain and imprecise references to real-world entities. The goal of entity resolution is to take a collection of uncertain entity references (or references, in short) from a single data source or multiple data sources, discover the unique set of underlying entities, and map each reference to its corresponding entity. This typically involves two subproblems – identification of references with different attributes to the same entity and disambiguation of references with identical attributes by assigning them to different entities.

Motivation and Background

Entity resolution is a common problem that comes up in different guises (and is given different names) in many computer science domains. Examples include computer vision, where we need to figure out when regions in two different images refer to the same underlying object (the correspondence problem), natural language processing when we would like to determine which noun phrases refer to the same underlying entity (co-reference resolution), and databases, where, when merging two databases or cleaning a database, we would like to determine when two tuple records are referring to the same real-world object (deduplication and data integration). Deduplication is important for removing redundancy and for accurate analysis. In information integration, determining approximate joins is important for consolidating information from multiple sources; most often there will not be a unique key that can be used to join tables across databases.

Such ambiguities in entity references can occur due to multiple reasons. Often times, data may have data entry errors, such as typographical errors. Multiple representations, such as abbreviations, are also possible. Different databases typically have different keys – one person database may use social security numbers, while another uses name and address.

Traditional entity resolution approaches focus on matching attributes of different references for resolving entities. However, many data sources have explicit or implicit relationships present among the entity references. These relations are indicative of relationships between the underlying entities themselves. For example, person records in census data are linked by family relationships such as sibling, parent, and spouse. Researchers collaborate mostly within their organization, or their research community, as a result of which references to related researchers tend to occur closely together. Recent entity resolution approaches in statistical relational learning make use of relationships between references to improve entity resolution accuracy and additionally to discover relationships between the underlying entities.

Theory/Solution

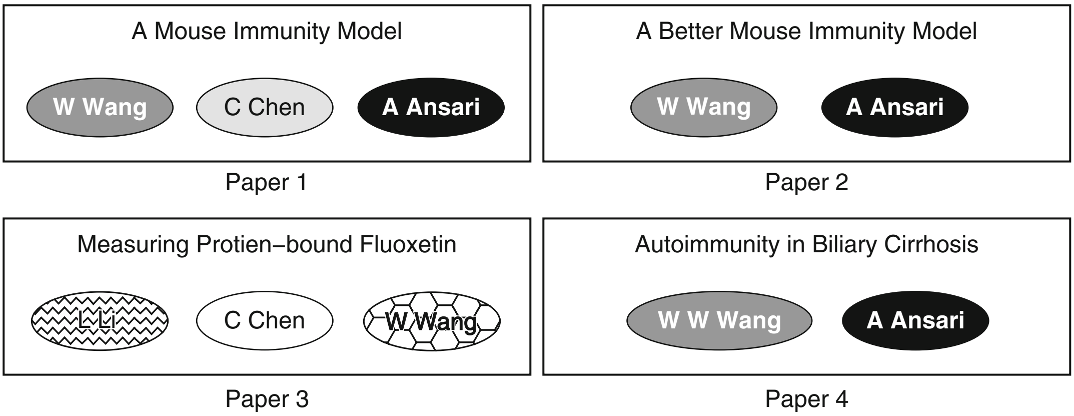
As an illustration of the entity resolution problem, consider the task of resolving the author references in a database of academic publications similar to DBLP, CiteSeer, or PubMed. Let us take as an example the following set of four papers:

1. W. Wang, C. Chen, and A. Ansari, "A mouse immunity model"
2. W. Wang and A. Ansari, "A better mouse immunity model"
3. L. Li, C. Chen, and W. Wang, "Measuring protein-bound fluoxetin"
4. W. W. Wang and A. Ansari, "Autoimmunity in biliary cirrhosis"

Now imagine that we would like to find out, given these four papers, which of these author names refer to the same author entities. This process involves determining whether paper 1 and paper 2 are written by the same author named Wang, or whether they are different authors. We need to answer similar questions about all such similar author names in the database.

In this example, it turns out there are six underlying author entities, which we will call *Wang*1 and *Wang*2, *Chen*1 and *Chen*2, *Ansari*, and *Li*. The three references with the name "A. Ansari" correspond to author $Ansari$ and the reference with name "L. Li" to author *Li*. However, the two references with name "C. Chen" map to two different authors *Chen*1 and *Chen*2. Similarly, the four references with name "W. Wang" or "W. W. Wang" map to two different authors. The "Wang" references from the first, second, and fourth papers correspond to author *Wang*1, while that from the third paper maps to a different author *Wang*2. This inference illustrates the twin problems of *identifying* "W. Wang" and "W. W. Wang" as the same author and *disambiguating* two references with name "W. Wang" as different authors. This is shown pictorially in Fig. 1, where references that correspond to the same authors are shaded identically. In the entity resolution process, all those and only those author references that are shaded identically should be resolved as corresponding to the same underlying entity.

Formally, in the entity resolution problem, we are given a set of references $\mathcal{R} = \{r_i\}$, where each reference r has attributes $r.A_1, r.A_2, \ldots, r.A_k$, such as observed names and affiliations for author references, as in our example above. The references correspond to some set of unknown entities $\mathcal{E} = \{e_i\}$. We introduce the notation $r.E$ to refer to the entity to which reference r corresponds. The goal is to recover the hidden set of entities $\mathcal{E} = \{e_i\}$ and the entity labels $r.E$ for individual references given the observed attributes of the references. In addition to the attributes, in some data sources we have information in the form of relationships between the references, such as coauthor relationships between author references in publication databases. We can capture the relationships with a set of hyper-edges $\mathcal{H} = \{h_i\}$. Each hyper-edge h may have attributes as well to capture the attributes of relationships, which we denote $h.A_1, h.A_2, \ldots, h.A_l$, and we use $h.R$ to denote the set of references that it connects. In our example, each rectangle denotes one hyper-edge corresponding to one paper in the database. The first hyper-edge corresponding to $Paper1$ has as its attribute the title "A mouse immunity model" and connects the three references having name attributes "W. Wang," "C. Chen," and "A. Ansari." A reference r can belong to zero or more hyper-edges, and we use $r.H$ to denote the set of hyper-edges in which r participates. For example, if we have paper, author, and venue references,



Entity Resolution, Fig. 1 The references in different papers in the bibliographic example. References to the same entity are identically shaded

then a paper reference may be connected to multiple author references and also to a venue reference. In general, the underlying references can refer to entities of different types, as in a publication database or in newspaper articles, which contain references to people, places, organizations, etc. When the type information is known for each reference, resolution decisions are restricted within references of the same type. Otherwise, the types may need to be discovered as well as part of the entity resolution process.

Traditional entity resolution approaches pose entity resolution as a pairwise decision problem over references based on their attribute similarity. It can also be posed as a ▶ graph clustering problem, where references are clustered together based on their attribute similarities and each cluster is taken to represent one underlying entity. Entity resolution approaches differ in how the similarities between references are defined and computed and how the resolution decisions are made based on these similarities. Traditionally, each pairwise decision is made independently of the others. For example, the decision to resolve the two *Wang* references from papers 1 and 3 would be made independently of the decision to resolve the two *Chen* references from the same papers.

The first improvement is to account for the similarity of the coauthor names when such relationships are available. However, this still does not consider the "entities" of the related references. For the two "Wang" references in the earlier example, the two "C. Chen" coauthors match regardless of whether they refer to *Chen1* or *Chen2*. The correct evidence to use here is that the "Chens" are not co-referent. In such a setting, in order to resolve the "W. Wang" references, it is necessary to *resolve* the "C Chen" references as well and not just consider their name similarity. In the collective relational entity resolution approach, resolutions are not made independently, but instead one resolution decision affects other resolutions via hyper-edges.

Below, we discuss the different entity resolution approaches in greater detail.

Attribute-Based Entity Resolution

As discussed earlier, exact matching of attributes does not suffice for entity resolution. Several sophisticated similarity measures have been developed for textual strings (Cohen et al. 2003; Chaudhuri et al. 2003) that may be used for unsupervised entity resolution. Finally, a weighted combination of the similarities over the different attributes for each reference is used to compute the attribute similarity between two references. An alternative is to use adaptive supervised algorithms that learn string ▶ similarity metrics from labeled data (Bilenko and Mooney 2003). In the traditional entity resolution approach (Fellegi and Sunter 1969; Cohen et al. 2003), similarity is computed for each pair of references r_i, r_j based on their attributes, and only those pairs that have similarity above some threshold are considered co-referent.

Efficiency

Even the attribute-only approach to entity resolution is known to be a hard problem computationally, since it is infeasible to compare all pairs of references using expensive similarity measures. Therefore, efficiency issues have long been a focus for data cleaning, the goal being the development of inexpensive algorithms for finding approximate solutions. The key mechanisms for doing this involve computing the matches efficiently and employing techniques commonly called "blocking" to quickly find potential duplicates (Hernández and Stolfo 1995; Monge and Elkan 1997), using cheap and index-based similarity computations to rule out non-duplicate pairs. Sampling approaches can quickly compute cosine similarity between tuples for fast text-joins within an SQL framework (Gravano et al. 2003). Error-tolerant indexes can also be used in data warehousing applications to efficiently look up a small but "probabilistically safe" set of reference tuples as candidates for matching for an incoming tuple (Chaudhuri et al. 2003). Generic entity resolution frameworks also exist for resolving and merging duplicates as a database operator and minimize the number of record-level and feature-level operations (Menestrina et al. 2006).

Probabilistic Models for Pairwise Resolution

The groundwork for posing entity resolution as a probabilistic ▸ classification problem was done by Fellegi and Sunter (1969), who studied the problem of labeling pairs of records from two different files to be merged as "match" (M) or "non-match" (U) on the basis of agreement γ among their different fields or attributes. Given an agreement pattern γ, the conditional probabilities $P(\gamma|M)$ and $P(\gamma|U)$ of γ given matches and non-matches are computed and compared to decide whether the two references are duplicates or not. Fellegi and Sunter showed that the probabilities $P(\gamma|M)$ and $P(\gamma|U)$ of field agreements can be estimated without requiring labeled training data if the different field agreements are assumed to be independent. Winkler (2002) used the EM algorithm to estimate the probabilities without making the independence assumption.

Probabilistic Models for Relational Entity Resolution

Probabilistic models that take into account interaction between different entity resolution decisions through hyper-edges have been proposed for named-entity recognition in natural language processing and for citation matching (McCallum and Wellner 2004; Singla and Domingos 2004). Such ▸ relational learning approaches introduce a decision variable y_{ij} for every pair of references r_i and r_j, but instead of inferring the y_{ij}'s independently, use conditional random fields for joint reasoning. For example, the decision variables for the "Wang" references and the "Chen" references in papers 1 and 3 would be connected to each other; features and functions would be defined to ensure that they are more likely to take up identical values.

Such relational models are supervised and require labeled data to train the parameters. One of the difficulties in using a supervised method for resolution is constructing a good training set that includes a representative collection of positive and negative examples. Accordingly, unsupervised relational models have also been developed (Li et al. 2005; Pasula et al. 2003; Bhattacharya and Getoor 2006). Instead of introducing pairwise decision variables, this category of approaches uses generative models for references using latent entity labels. Note that, here, the number of entities is unknown and needs to be discovered automatically from the available references. Relationships between the references, such as co-mentions or co-occurrences, are captured using joint distributions over the entity labels.

All of these probabilistic models have been shown to perform well in practice and have the advantage that the match/non-match decisions do not depend on any user-specified similarity measures and thresholds but are learned directly from data. However, this benefit comes at a price. Inference in relational probabilistic models is an expensive process. Exact inference is mostly intractable and approximate strategies such as loopy belief propagation and Monte Carlo sampling strategies are employed. Even these approximate strategies take several iterations to converge, and extending such approaches to large datasets is still an open problem.

Other Approaches for Relational Entity Resolution

Alternative approaches (Bhattacharya and Getoor 2007; Kalashnikov et al. 2005; Dong et al. 2005) consider relational structure of the entities for data integration but avoid the complexity of probabilistic inference. By avoiding a formal probabilistic model, these approaches can handle complex and longer-range relationships between different entity references, and the resolution process is significantly faster as well. Such approaches also create pairwise decision nodes between references and create a dependency graph over them to capture the relationships in the data. But instead of performing probabilistic inference, they keep updating the value associated with each decision node by propagating relational evidence from one decision node to another over the dependency graph.

When the relationships between the references and the entities can be captured in a single graph, the matching entity for a specific reference may be identified using path-based similarities between their corresponding nodes in the graph. The connection strength associated with each edge in the graph can be determined in the unsupervised fashion given all the references, their candidate entity choices, and the relationships between them, by solving a set of nonlinear equations (Kalashnikov et al. 2005). This approach is useful for incremental data cleaning when the set of entities currently in the database is known and an incoming reference needs to be matched with one of these entities.

An alternative approach to performing collective entity resolution using relational evidence is to perform collective relational clustering (Bhattacharya and Getoor 2007). The goal here is to cluster the references into entities by taking into account the relationships between the references. This is achieved by defining a similarity measure between two clusters of references that take into account not only the attribute similarity of the references in the two clusters but also the neighboring clusters of each cluster. The neighboring clusters of any reference cluster c are defined by considering the references r' connected to references r belonging to c via hyper-edges and the clusters to which these related references belong. If the $r.C$ represents the current cluster for reference c, then $N(c) = \bigcup r'.C$, where $r.H = r'.H$ and $r.C = c$. For instance, the neighboring clusters for a $Wang$ cluster in our example containing the $Wang$ references from papers 1, 2, and 4 are the $Ansari$ cluster and the $Chen$ clusters containing the other references from the same papers. The relational similarity between two clusters is then computed by comparing their neighborhoods. This relational similarity complements attribute similarity in the combined similarity between two clusters. Intuitively, two entities are likely to be the same if they are similar in attributes and are additionally connected to the same other entities. Collective relational clustering can be efficiently implemented by maintaining a priority queue for merge-able cluster pairs and updating the "neighboring" queue elements with every merge operation.

Applications

Data cleaning and reference disambiguation approaches have been applied and evaluated in a number of domains. The earliest applications were on medical data. Census data is an area where detection of duplicates poses a significant challenge and Winkler (2002) has successfully applied his research and other baselines to this domain. A great deal of work has been done making use of bibliographic data (Pasula et al. 2003; Singla and Domingos 2004; Bhattacharya and Getoor 2007). Almost without exception, the focus has been on the matching of citations. Work in co-reference resolution and disambiguating entity mentions in natural language processing (McCallum and Wellner 2004) has been applied to text corpora and newswire articles like the TREC corpus. There have also been significant applications in information integration in data warehouses (Chaudhuri et al. 2003).

Cross-References

▸ Classification
▸ Data Preparation
▸ Graph Clustering
▸ Record Linkage
▸ Similarity Measures
▸ Statistical Relational Learning

Recommended Reading

Bhattacharya I, Getoor L (2006) A latent dirichlet model for unsupervised entity resolution. In: The SIAM international conference on data mining (SIAM-SDM), Bethesda

Bhattacharya I, Getoor L (2007) Collective entity resolution in relational data. ACM Trans Knowl Discov Data 1(1):5

Bilenko M, Mooney RJ (2003) Adaptive duplicate detection using learnable string similarity measures. In: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining (KDD-2003), Washington, DC

Chaudhuri S, Ganjam K, Ganti V, Motwani R (2003) Robust and efficient fuzzy match for online data cleaning. In: Proceedings of the 2003 ACM SIGMOD international conference on management of data, San Diego, pp 313–324

Cohen WW, Ravikumar P, Fienberg SE (2003) A comparison of string distance metrics for name-matching tasks. In: Proceedings of the IJCAI-2003 workshop on information integration on the web, Acapulco, pp 73–78

Dong X, Halevy A, Madhavan J (2005) Reference reconciliation in complex information spaces. In: The ACM international conference on management of data (SIGMOD), Baltimore

Fellegi IP, Sunter AB (1969) A theory for record linkage. J Am Stat Assoc 64:1183–1210

Gravano L, Ipeirotis P, Koudas N, Srivastava D (2003) Text joins for data cleansing and integration in an rdbms. In: 19th IEEE international conference on data engineering, Bangalore

Hernández MA, Stolfo SJ (1995) The merge/purge problem for large databases. In: Proceedings of the 1995 ACM SIGMOD international conference on management of data (SIGMOD-95), San Jose, pp 127–138

Kalashnikov DV, Mehrotra S, Chen Z (2005) Exploiting relationships for domain-independent data cleaning. In: SIAM international conference on data mining (SIAM SDM), Newport Beach, 21–23 Apr 2005

Li X, Morie P, Roth D (2005) Semantic integration in text: from ambiguous names to identifiable entities. AI Mag Spec Issue Semant Integr 26(1):45–58

McCallum A, Wellner B (2004) Conditional models of identity uncertainty with application to noun coreference. In: NIPS, Vancouver

Menestrina D, Benjelloun O, Garcia-Molina H (2006) Generic entity resolution with data confidences. In: First Int'l VLDB workshop on clean databases, Seoul

Monge AE, Elkan CP (1997) An efficient domain-independent algorithm for detecting approximately duplicate database records. In: Proceedings of the SIGMOD 1997 workshop on research issues on data mining and knowledge discovery, Tuscon, pp 23–29

Pasula H, Marthi B, Milch B, Russell S, Shpitser I (2003) Identity uncertainty and citation matching. In: Advances in neural information processing systems 15, Vancouver. MIT, Cambridge

Singla P, Domingos P (2004) Multi-relational record linkage. In: Proceedings of 3rd workshop on multi-relational data mining at ACM SI GKDD, Seattle

Winkler WE (2002) Methods for record linkage and Bayesian networks. Technical report, Statistical Research Division, U.S. Census Bureau, Washington, DC

EP

▸ Expectation Propagation

Epsilon Cover

Thomas Zeugmann
Hokkaido University, Sapparo, Japan

Motivation and Background

Epsilon covers were introduced in calculus. So we provide here a very general definition.

Definition

Let (M, ϱ) be a metric space, let $S \subseteq M$, and let $\varepsilon > 0$. A set $E \subseteq M$ is an *ε-cover* for S, if for every $s \in S$ there is an $e \in E$ such that $\varrho(s, e) \leq \varepsilon$.

An *ε-cover* E is said to be *proper*, if $E \subseteq S$.

Application

The notion of an ε-cover is frequently used in kernel-based learning methods.

For further information, we refer the reader to Herbrich (2002).

Cross-References

- ▸ Statistical Machine Translation
- ▸ Support Vector Machines

Recommended Reading

Herbrich R (2002) Learning kernel classifiers: theory and algorithms. MIT, Cambridge

Epsilon Nets

Thomas Zeugmann
Hokkaido University, Sapparo, Japan

Motivation and Background

Epsilon nets were introduced by Haussler and Welz (1987), and their usefulness for computational learning theory has been discovered by Blumer et al. (1989).

Let $X \neq \emptyset$ be any learning domain and let $\mathcal{C} \subseteq \wp(X)$ be any nonempty concept class. For the sake of simplicity, we also use $\mathcal{C}$ here as hypothesis space. In order to guarantee that all probabilities considered below do exist, we restrict ourselves to *well-behaved* concept classes (see ▸ PAC Learning).

Furthermore, let D be any arbitrarily fixed probability distribution over the learning domain X and let $c \in \mathcal{C}$ be any fixed concept.

A hypothesis $h \in \mathcal{C}$ is said to be *bad* for c iff

$$d(c, h) = \sum_{x \in c \triangle h} D(x) > \varepsilon.$$

Furthermore, we use

$$\Delta(c) =_{df} \{h \triangle c \mid h \in \mathcal{C}\}$$

to denote the set of all possible *error regions* of c with respect to $\mathcal{C}$ and D. Moreover, let

$$\Delta_\varepsilon(c) =_{df} \{h \triangle c \mid h \in \mathcal{C},\ d(c, h) > \varepsilon\}$$

denote the set of all *bad error regions* of c with respect to $\mathcal{C}$ and D.

Now we are ready to formally define the notion of an ε-net.

Definition

Let $\varepsilon \in (0, 1)$, and let $S \subseteq X$. The set S is said to be an *ε-net for* $\Delta(c)$ iff $S \cap r \neq \emptyset$ for all $r \in \Delta_\varepsilon(c)$.

Remarks

Conceptually, a set S constitutes an ε-net for $\Delta(c)$ iff every bad error region is hit by at least one point in S.

Example

Consider the one-dimensional Euclidean space $\mathbb{E}$, and let $X = [0, 1] \subseteq \mathbb{E}$. Furthermore, let $\mathcal{C}$ be the set of all closed intervals $[a, b] \subseteq [0, 1]$. Consider any fixed $c \in \mathcal{C}$, and let D be the uniform distribution, i.e., $D([a, b]) = 1/(b - a)$ for all $[a, b] \in \mathcal{C}$. Furthermore, let $h \in \mathcal{C}$; then we may write $c \triangle h = I_1 \cup I_2$, where $I_1, I_2 \in \mathcal{C}$. Let $\varepsilon \in (0, 1)$ be arbitrarily fixed, and let

$$S = \{k\varepsilon/2 \mid 0 \le k \le \lceil 2/\varepsilon \rceil,\ k \in \mathbb{N}\}\,.$$

Then, S forms an ε-net for $\Delta(c)$. This can be seen as follows. Assume $r \in \Delta_\varepsilon(c)$. Then, $D(I_1) > \varepsilon/2$ or $D(I_2) > \varepsilon/2$. Now, by the definition of S, it is obvious that $D(I_i) > \varepsilon/2$ implies $I_i \cap S \ne \emptyset$, $i = 1, 2$.

Application

Recall that in ▸ PAC Learning, the general strategy to design a learner has been to draw a sufficiently large finite sample and then to find a hypothesis that is consistent with it. For showing that this strategy is always successful, the notion of an ε-net plays an important role. This can be expressed by the following observation.

Observation. Let $S = \{x_1, \ldots, x_m\}$ be an ε-net for $\Delta(c)$, and let $h \in \mathcal{C}$ be any hypothesis such that $h(x_i) = c(x_i)$ for all $1 \le i \le m$, i.e., h is consistent. Then we have $d(c, h) \le \varepsilon$.

It then remains to show that the ▸ VC Dimension of $\mathcal{C}$ and of $\Delta(c)$ are the same and to apply Sauer's lemma to complete the proof.

For further information, we refer the reader to Blumer et al. (1989) as well as to Kearns and Vazirani (1994).

Cross-References

▸ PAC Learning
▸ VC Dimension

Recommended Reading

Blumer A, Ehrenfeucht A, Haussler D, Warmuth MK (1989) Learnability and the Vapnik-Chervonenkis dimension. J ACM 36(4):929–965

Haussler D, Welz E (1987) Epsilon nets and simplex range queries. Discret & Comput Geom 2:127–151 (1987)

Kearns MJ, Vazirani UV (1994) An introduction to computational learning theory. MIT, Cambridge

Equation Discovery

Ljupčo Todorovski
University of Ljubljana, Ljubljana, Slovenia

Synonyms

Computational discovery of quantitative laws; Symbolic regression

Definition

Equation discovery is a machine learning task that deals with the problem of learning quantitative laws and models, expressed in the form of equations, in collections of measured numeric data. Equation discovery methods take at input a ▸ data set consisting of measured values of a set of numeric variables of an observed system or phenomenon. At output, equation discovery methods provide a set of equations, such that, when used to calculate the values of system variables, the calculated values closely match the measured ones.

Motivation and Background

Equation discovery methods can be used to solve complex modeling tasks, i.e., establishing a mathematical model of an observed system. Modeling tasks are omnipresent in many scientific and engineering domains.

Equation discovery is strongly related to *system identification*, another approach to mathematical modeling. System identification methods work under the assumption that the structure of the model (the form of the model equations) is known or comes from a well-defined class of model structures, such as polynomials or neural networks. Therefore, they are mainly concerned with the parameter estimation task, that is, the task of determining the values of the model parameters that minimize the discrepancy between measured data and data obtained by simulating the model. Equation discovery methods, on the other hand, aim at identifying both, an adequate structure of the model equations and appropriate values of the model parameters.

▶ Regression also deals with building predictive models from numeric data. The focus of regression methods is on building descriptive black-box models that can reconstruct the training data with high accuracy. In contrast, equation discovery methods focus on establishing explanatory models that, beside accurate predictions, provide explanations of the mechanisms that govern the behavior of the modeled system.

Early equation discovery methods dealt with rediscovering empirical laws from the history of science (this is where the synonym "computational discovery of quantitative laws" comes from). Through the years, the focus of the equation discovery methods has shifted from discovering quantitative laws to modeling real-world systems.

Structure of the Learning System

The task of equation discovery can be decomposed into two closely coupled subtasks of structural identification and parameter estimation. The first task of structural identification deals with the problem of finding the optimal structure of an equation. The second task of parameter estimation deals with the problem of finding the optimal values of the constant parameters in the equation. General approaches to and specific methods for equation discovery use different techniques to solve these two subtasks.

Approaches and Methods

There are two general and fundamentally different approaches to equation discovery. The first approach relies on a definition of a space of candidate equation structures. Following this definition, a generate-and-test (or ▶ learning as search) approach is used to generate different equation structures, solve the parameter estimation task for each of them, and report those equations that most closely approximate the data. The second approach relies on heuristics, used by scientists and engineers in the discovery or modeling processes, to establish an appropriate equation structure.

The first equation discovery system, Bacon (Langley 1981), follows the second approach described above. It incorporates a set of data-driven heuristics for detecting regularities (constancies and trends) in measured data and for formulating hypotheses based on them. An example heuristic would, when faced with a situation where the values of two observed variables increase/decrease simultaneously, introduce a new equation term by multiplying them. Furthermore, Bacon builds equation structure at different levels of description. At each level of description, all but two variables are held constant and hypotheses connecting the two changing variables are considered. Using a relatively small set of data-driven heuristics, Bacon is able to rediscover a number of physical laws including the ideal gas law, the law of gravitation, the law of refraction, and Black's specific heat law.

An alternative set of heuristics for equation discovery can be derived from dimensional analysis that is routinely used to check the plausibility of equations by using rules that specify the proper ways to combine variables and terms with different *measurements units*, different measurement scales, or types thereof. Following these rules, equation discovery method Coper (Kokar 1986) considers only equation structures that properly combine variables and constants, given the knowledge about their exact measurement units. Equation discovery method SDS (Takashi and Hiroshi 1998) extends Coper to cases, where the exact measurement units of the variables and constants involved in the equation are not known, but only knowledge about the types of the ▶ measurement scales is available.

Finally, the heuristics and design of the equation discovery method E* (Schaffer 1993) is based on a systematic survey of more than a hundred laws and models published in the Physical Review journal. The review shows that many of the published laws and models follow one of five different equation structures. By including only these five structures as its main heuristic for solving the structure identification task (implementing it as a ▶ language bias), E* was able to reconstruct the correct laws and models in about a third of the test cases collected from the same journal.

Abacus (Falkenhainer and Michalski 1990) was the first equation discovery method that followed the generate-and-test (or ▸ learning as search) approach, mentioned above. Abacus experimented with different search strategies within a fixed space of candidate equation structures. Other methods that follow the generate-and-test approach differ in the ways they define the space of candidate equation structures and solve the parameter estimation task.

Equation discovery methods EF (Zembowitz and Zytkow 1992) and Lagrange (Džeroski and Todorovski 1995) explore the space of polynomial equation structures that are linear in the constant parameters, so they apply ▸ linear regression to estimate parameters. The user can shape the space of candidate structures by specifying parameters, such as, the maximal polynomial degree, the maximal number of multiplicative terms included in a polynomial, and a set of functions that can be used to transform the original variables before combining them into multiplicative terms.

While all of the above methods assume a fixed predefined ▸ language bias (via specification of the class of candidate equation structures or via heuristics for establishing appropriate structure), equation discovery method Lagramge (Todorovski and Džeroski 1997) employs dynamic declarative ▸ language bias, that is, let the user of the equation discovery method choose or specify the space of candidate equation structures. In its first version, Lagramge uses the formalism of context-free grammars for specifying the space of equation structures. The formalism has been shown to be general enough to allow users to build their specification upon many different types of modeling knowledge, from measurement units to very specific knowledge about building models in a particular domain of interest (Todorovski and Džeroski 2007). For solving the structure identification task, Lagramge defines a refinement operator that orders the search space of candidate equation structures, defined by the user-specified grammar, from the simplest ones to more complex. Exhaustive and ▸ beam search strategies are then being employed to the search space and for each structure considered during the search, Lagramge uses gradient-descent methods for nonlinear optimization to solve the parameter estimation task. The heuristic function that guides the search is based on the ▸ mean squared error that measures the discrepancy between the measured and simulated values of the observed system variables. Alternatively, Lagramge can use heuristic function that takes into account the complexity of the equation and is based on the ▸ minimum description length principle.

Successors of Lagramge, equation discovery methods, Lagramge 2 (Todorovski and Džeroski 2007), IPM (Bridewell et al. 2008), and HIPM (Todorovski et al. 2005), primarily focus on the improvement of the knowledge representation formalism used to formalize the modeling knowledge and transform it to ▸ language bias for equation discovery. All of them follow the paradigm of ▸ inductive process modeling.

Types of Equations

At first, equation discovery methods dealt with the problem of learning algebraic equations from data. Equation discovery method Lagrange (Džeroski and Todorovski 1995) extended the scope of equation discovery to modeling dynamics from ▸ time series data with ordinary differential equations. It took a naïve approach based on transforming the task of discovering ordinary differential equations to the simpler task of discovering algebraic equations, by extending the set of observed system variables with numerically calculated time derivatives thereof. By doing so, any of the existing equation discovery methods could be, in principle, used to discover differential equations. However, the naïve approach has a major drawback of introducing large numerical errors, due to instability of methods for numerical differentiation. Equation discovery method GoldHorn (Križman et al. 1995) replaced the instable numerical differentiation with the stable numerical methods for the inverse problem of integration. Goldhorn also upgrades Lagrange with filtering methods to cope with measurement errors and noisy data.

While ordinary differential equations can model systems that change their state along a single dimension, time, partial differential equations can be used to model systems that change along many (temporal and spatial) dimensions. The naïve approach of introducing numerically calculated partial derivatives has been used in the Paddles (Todorovski et al. 2000) method for discovery of partial differential equations. The method first slices the measurement data into narrow spatial subsets, induces ordinary differential equations in each of them, and uses most frequently obtained equation structures to extend them with partial derivatives and to obtain a relatively small class of partial differential equation structures to explore. All the equation discovery tasks in Paddles are solved using Lagramge (Todorovski and Džeroski 1997).

Applications

Equation discovery methods have been applied to various tasks of discovering equation-based laws and models from measured and/or simulation data. Application domains range from physics (mechanical and electrical engineering, fluid dynamics) (Takashi and Hiroshi 1998; Todorovski and Džeroski 1997, 2007), through ecology (population dynamics) (Todorovski and Džeroski 2007; Todorovski et al. 2005) to biochemistry (chemical kinetics) (Džeroski and Todorovski 2008; Langley et al. 2006).

Cross-References

- ▶ Identification
- ▶ Inductive Process Modeling
- ▶ Language Bias
- ▶ Learning as Search
- ▶ Linear Regression
- ▶ Measurement Scales
- ▶ Regression

Recommended Reading

Bridewell W, Langley P, Todorovski L, Džeroski S (2008) Inductive process modeling. Mach Learn 71(1):1–32

Džeroski S, Todorovski L (1995) Discovering dynamics: from inductive logic programming to machine discovery. J Intell Inf Syst 4(1): 89–108

Džeroski S, Todorovski L (2008) Equation discovery for systems biology: finding the structure and dynamics of biological networks from time course data. Curr Opin Biotechnol 19:1–9

Falkenhainer B, Michalski R (1990) Integrating quantitative and qualitative discovery in the ABACUS system. In: Kodratoff Y, Michalski R (eds) Machine learning: an artificial intelligence approach. Morgan Kaufmann, San Mateo

Kokar MM (1986) Determining arguments of invariant functional descriptions. Mach Learn 1(4): 403–422

Križman V, Džeroski S, Kompare B (1995) Discovering dynamics from measured data. Electrotech Rev 62(3–4):191–198

Langley P (1981) Data-driven discovery of physical laws. Cogn Sci 5(1):31–54

Langley P, Shiran O, Shrager J, Todorovski L, Pohorille A (2006) Constructing explanatory process models from biological data and knowledge. Artif Intell Med 37(3):191–201

Schaffer C (1993) Bivariate scientific function finding in a sampled, real-data testbed. Mach Learn 12(1–3):167–183

Takashi W, Hiroshi M (1998) Discovery of first-principle equations based on scale-type-based and data-driven reasoning. Knowl-Based Syst 10(7):403–411

Todorovski L, Bridewell W, Shiran O, Langley P (2005) Inducing hierarchical process models in dynamic domains. In: Veloso MM, Kambhampati S (eds) Proceedings of the twentieth national conference on artificial intelligence, Pittsburgh

Todorovski L, Džeroski S (1997) Declarative bias in equation discovery. In: Fisher DH (ed) Proceedings of the fourteenth international conference on machine learning, Nashville

Todorovski L, Džeroski S (2007) Integrating domain knowledge in equation discovery. In Džeroski S, Todorovski L (eds) Computational discovery of scientific knowledge. LNCS, vol 4660. Springer, Berlin

Todorovski L, Džeroski S, Srinivasan A, Whiteley J, Gavaghan D (2000) Discovering the structure of partial differential equations from example behaviour. In: Langley P (ed) Proceedings of the seventeenth international conference on machine learning, Stanford

Zembowitz R, Zytkow J (1992) Discovery of equations: experimental evaluation of convergence. In: Swartout WR (ed) Proceedings of the tenth national conference on artificial intelligence, San Jose

Error

▶ Error Rate

Error Correcting Output Codes

Synonyms

▶ ECOC

Definition

Error correcting output codes are an ▶ ensemble learning technique. It is applied to a problem with multiple classes, decomposing it into several binary problems. Each class is first encoded as a binary string of length T, assuming we have T models in the ensemble. Each model then tries to separate a subset of the original classes from all the others. For example, one model might learn to distinguish "class A" from "not class A." After the predictions, with T models we have a binary string of length T. The class encoding that is closest to this binary string (using Hamming distance) is the final decision of the ensemble.

Recommended Reading

Kong EB, Dietterich TG (1995) Error-correcting output coding corrects bias and variance. In: International conference on machine learning, Tahoe City

Error Curve

▶ Learning Curves in Machine Learning

Error Rate

Kai Ming Ting
Federation University, Mount Helen, VIC, Australia

Synonyms

Error

Definition

Error rate refers to a measure of the degree of prediction error of a model made with respect to the true model.

The term *error rate* is often applied in the context of ▶ classification models. In this context, *error rate* $= \mathrm{P}(\lambda(X) \neq Y)$, where XY is a joint distribution and the classification model λ is a function $X \rightarrow Y$. Sometimes this quantity is expressed as a percentage rather than a value between 0.0 and 1.0.

The error rate of a model is often assessed or estimated by applying it to ▶ test data for which the class labels (Y values) are known. The error rate of a classifier on test data may be calculated as *number of incorrectly classified objects/total number of objects*. Alternatively, a smoothing function may be applied, such as a ▶ Laplace estimate or an *m*-estimate.

Error rate is directly related to ▶ accuracy, such that *error rate* $= 1.0 -$ *accuracy* (or when expressed as a percentage, *error rate* $= 100 -$ *accuracy*).

Two common measures of *error rate* for ▶ regression models are ▶ mean squared error and ▶ mean absolute error.

Cross-References

▶ Accuracy
▶ Confusion Matrix
▶ Mean Absolute Error
▶ Mean Squared Error

Error Squared

Synonyms

▸ Squared error

Definition

Error squared is a common ▸ loss function used with ▸ regression. This is the square of the difference between the predicted and true values.

Error-Correcting Output Codes (ECOC)

▸ Class Binarization

Estimation of Density Level Sets

▸ Density-Based Clustering

Evaluation

Evaluation is a process that assesses some property of an artifact. In machine learning, two types of\break artifacts are most commonly evaluated, models and {algorithms}. ▸ Model evaluation often focuses on the predictive efficacy of the model, but may also assess factors such as its complexity, the ease with which it can be understood, or the computational requirements for its application. ▸ Algorithm evaluation often focuses on evaluation of the models an algorithm produces, but may also appraise its computational efficiency.

Evaluation Data

▸ Test Data
▸ Test Set

Evaluation of Learning Algorithms

Geoffrey I. Webb
Faculty of Information Technology, Monash University, Victoria, Australia

Abstract

It is often desirable to assess the properties of a learning algorithm. Frequently such evaluation take the form of comparing the relative suitability of a set of algorithms for a specific task or class of tasks. Learning algorithm evaluation is the process of performing such assessment of a learning algorithm.

Synonyms

Algorithm Evaluation; Learning Algorithm Evaluation

Definition

Learning algorithm evaluation is the process of assessing a property or properties of a learning algorithm.

Motivation and Background

It is often valuable to assess the efficacy of a learning algorithm. In many cases, such assessment is relative, that is, evaluating which of several alternative algorithms is best suited to a specific application.

Processes and Techniques

Many learning algorithms have been proposed. In order to understand the relative merits of these alternatives, it is necessary to evaluate them. The primary approaches to evaluation can be characterized as either theoretical or experimental. Theoretical evaluation uses formal methods to infer properties of the algorithm, such as its computational complexity (Papadimitriou 1994), and also employs the tools of computational learning theory to assess learning theoretic properties. Experimental evaluation applies the algorithm to learning tasks in order to study its performance in practice.

There are many different types of property that may be relevant to assess depending upon the intended application. These include algorithmic properties, such a time and space complexity. These algorithmic properties are often assessed separately with respect to performance when learning a model, that is, at *training time*, and performance when applying a learned model, that is, at *test time*.

Other types of property that are often studied are the properties of the models that are learned (see ▸ model evaluation). Strictly speaking, such properties should be assessed with respect to a specific application or class of applications. However, much machine learning research includes experimental studies in which algorithms are compared using a set of data sets with little or no consideration given to what class of applications those data sets might represent. It is dangerous to draw general conclusions about relative performance on any application from relative performance on such a sample of some unknown class of applications. Such experimental evaluation has become known disparagingly as a *bake-off*.

An approach to experimental evaluation that may be less subject to the limitations of bake-offs is the use of experimental evaluation to assess a learning algorithm's bias and variance profile. Bias and variance measure properties of an algorithm's propensities in learning models rather than being directly properties of the models that are learned. Hence they may provide more general insights into the relative characteristics of alternative algorithms than do assessments of the performance of learned models on a finite number of applications. One example of such use of bias-variance analysis is found in Webb (2000).

Techniques for experimental algorithm evaluation include bootstrap sampling, cross validation, and holdout evaluation.

Cross-References

▸ Model Evaluation

Recommended Reading

Hastie T, Tibshirani R, Friedman J (2001) The elements of statistical learning. Springer, New York

Mitchell TM (1997) Machine learning. McGraw-Hill, New York

Papadimitriou CH (1994) Computational complexity. Addison-Wesley, Reading

Webb GI (2000) MultiBoosting: a technique for combining boosting and wagging. Mach Learn 40(2):159–196

Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, Amsterdam/Boston

Evaluation of Model Performance

▸ Model Evaluation

Evaluation Set

▸ Test Set

Event Extraction from Media Texts

Gregor Leban[1], Blaž Fortuna[1], and
Marko Grobelnik[2]
[1]Jozef Stefan Institute, Ljubljana, Slovenia
[2]Artificial Intelligence Laboratory, Jožef Stefan Insitute, Ljubljana, Slovenia

Abstract

The chapter describes the topic of using news content to automatically detect world events mentioned in the news. Various tasks required for identifying events are presented, such as semantic annotation, article clustering and

cross-lingual cluster matching. Given the identified events we also describe how date, location, relevant entities and other core event details can be determined automatically.

Definition

Event extraction from text is an area of research that involves identifying mentions of significant world events described in media documents, such as news articles. The goal is to identify the world events and extract as much information as possible about them in a structured form. Ideally, the extracted information should contain details about what happened, when, where, and who was involved in the event. Since relevant events are reported in numerous articles, the methods for detection of events can exploit this fact when identifying events and extracting event properties.

Motivation and Background

News outlets produce large amounts of news content every day. Most of the news articles describe recent happenings in the world, such as meetings of important politicians, natural disasters, societal issues, sports events, or pastimes of celebrities. The importance of the generated news content varies significantly – news about an approaching hurricane can be considered as much more important and relevant than a news article about a party held by a local politician. For the purposes of this paper, we will call important happenings *events*. There is no objective way to distinguish between important and non-important news stories, but a practical approach that can be used is to treat news as important if it is being reported by several news publishers. For practical purposes we can therefore define an event as a happening that is being covered in news articles by several news publishers.

News articles are written in a natural language which makes them easy to understand by humans, but hard to process by computers. Understanding information being described in an article requires the use of common sense, common knowledge, implicit information, and knowledge on how to disambiguate. Since it's hard to extract knowledge from the articles, it is also difficult to perform accurate information retrieval. Imagine, for example, that you would like to learn from the news about the events that happened in Washington state in the last month. The word "Washington" is for the computer just a sequence of letters. One can use it to perform a keyword search, which will however return various articles – from the ones about the state of Washington, about any of the 40 cities names Washington, as well as numerous people who are also named Washington. Even if all articles would be relevant, they are not grouped – there would be tens or hundreds of articles describing the same event, and it would be up to the reader to find if an article describes some event you have already seen or not. Additionally, there would also be no summary of what the event was about – the reader would have to read the articles and learn about that himself.

To make learning about the events a more pleasant experience, we would like to convert the unstructured information expressed in news articles into a structured form that can be stored in a machine-readable way. This is not a trivial task and requires several steps of processing. These steps include syntactic analysis, semantic enrichment, entity linking, document clustering, and information extraction. The final result of the processing is a structured database of world events. Due to the extensive metadata, it is possible to find events based on the date, location, or relevant entities. Articles about an event are grouped together which helps significantly to reduce the information overload. A summary of an event can also be obtained by aggregating common information from multiple news articles.

To our knowledge, there are at least three systems that are identifying world events by analyzing news media. GDELT project (Gao et al. 2013; Leetaru et al. 2013) performs event detection by extracting information from individual sentences in the news articles. Since several events are potentially extracted from a single article, it contains a huge collection of events (over 200 million) that were extracted from 1979 to the present. European media monitor (Steinberger et al. 2005; Pouliquen et al. 2008) focuses on the identification of current political events by combining and processing news articles in multiple languages. Event Registry (Leban et al. 2014a,b) similarly extracts world events with the additional metadata from articles in several languages. In the next sections, we will describe some of the core components that are needed by these systems.

Structure of Learning System

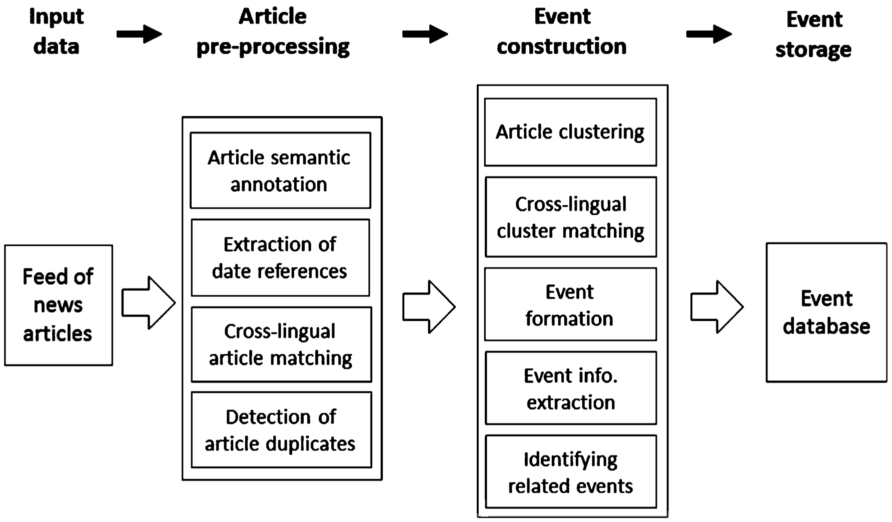
The identification of events from news articles requires a pipeline of services or components that provide specific functionalities. These services are shown in the Fig. 1 and will be described next.

News Collection

The first step in identification of events from news is to obtain the news content from a large set of news publishers. The content can be collected by either crawling the website of the news publishers or by identifying their RSS feeds and extracting article information from them. The use of RSS feeds, which are almost always available, is a better approach since they are significantly less data and time intensive compared to repeatedly crawling the whole websites. The RSS feeds do however often contain only article excerpts and crawling of the article page is therefore still needed. The main technical challenge with crawling the page is the identification of the article content and removal of the rest of the page. It is also important to extract as much metadata about the article as possible. This metadata can include the title of the article, publisher's name, date of publishing, author, etc.

Text Annotation

The collected news articles contain just plain text – there is no semantic information available about its content. In order to be able to extract semantic information about the described event, the text first needs to be annotated with semantic information that can be detected in the text. Common types of annotations are the named entities (people, locations, organizations) mentioned in the text and article topics. The challenge in annotation is twofold: first, the token or phrase that represents a named entity (such as "Paris") has to be identified, and second, it needs to be linked to a resource identifier that semantically represents the entity (such as a URI in a knowledge base). The first task is called


Input data
Article pre-processing
Event construction
Event storage
Feed of news articles
Article semantic annotation
Extraction of date references
Cross-lingual article matching
Detection of article duplicates
Article clustering
Cross-lingual cluster matching
Event formation
Event info. extraction
Identifying related events
Event database

Event Extraction from Media Texts, Fig. 1 Various components required in the process of extracting events from news articles named entity recognition and can be best solved using conditional random fields or more recently with convolutional neural networks. The second task is called entity linking and requires the use of a knowledge base containing an extensive set of relevant entities. Systems for entity linking such as Wikipedia Miner (Milne and Witten 2013) rely on the use of open knowledge bases such as DBpedia, Freebase, or YAGO.

An important type of data annotations are also temporal expressions mentioned in the text. Their detection is crucial in order to determine the date of the event that is being described in the news. Dates can be expressed in an absolute ("July 15th, 2015", "2015-07-12") or relative form ("yesterday morning," "last week"). The detection of absolute temporal expressions can be efficiently performed using a set of regular expressions. The relative expressions can either be identified using rule-based or sequence labeling approaches. The detected relative expressions can then be normalized into the absolute form using the article's publish date.

Clustering Approach to Event Identification

In order to identify events, a clustering approach can then be applied in order to group similar articles that describe the same event. The reasoning behind using clustering is the reasonable assumption that if articles are describing the same event, they will share similar vocabulary and mention similar entities. The most valuable features for the clustering algorithm are therefore the article text itself as well as the detected named entities and topics. The article text can be transformed into the bag-of-words form where each term in the document is normalized according to a chosen weighting scheme, such as TF-IDF. A feature vector can be generated for each article by concatenating the weights of the article terms and the mentioned named entities. A similarity measure, such as cosine similarity, can then be used to compute the similarity between individual articles.

Given the article feature vectors and the similarity measure, the clusters representing events can be identified using various clustering methods. European media monitor, for example, uses an agglomerative bottom-up clustering algorithm to group all articles published in a 24-h time window. Articles are grouped into the same cluster as long as their similarity is above the selected threshold. Centroid vectors of the obtained clusters are also compared with clusters identified on a previous day. The clusters that are found to be similar enough are merged and therefore represented as a single event. This allows the method to identify events that span across several days.

Event registry, on the other hand, uses an online approach to clustering. Each new article is clustered immediately after being added to the system. The clustering approach works as follows. The feature vector of the article is first being compared to the feature vectors of the centroids of all existing clusters. If the cosine similarity to the most similar centroid is above the selected threshold, the article is simply assigned to the cluster. Otherwise, a new, so-called *micro-cluster* is created containing only the single article. As new articles are added to the system, the micro-clusters can grow in size as articles are being added to them. Once they reach a certain size (depending on the language, this can be between three and ten articles), they are no longer considered as micro-clusters but instead as proper events. Micro-clusters that never reach the necessary size are not considered as events and are eventually removed. There are also different validation methods that are being called regularly in order to ensure the highest quality of the clusters. As clusters grow, for example, it can occur that the centroids of two clusters become more and more similar. One of the validation methods therefore checks different pairs of clusters and merges them in case their similarity, as measured by the cosine similarity of their centroid vectors, is above the threshold. Additionally, a separate method also checks each cluster if it is still sufficiently coherent or should instead be split into two separate clusters. The main idea behind splitting is to project all articles in the cluster onto a line and divide them into two groups depending on whether their projection was left or right of the centroid. This is repeated several times. In the first iteration the first principal component of the original cluster is used as the projection line. In the following steps, the articles are projected onto the line that passes the centroids of the two groups obtained in the previous iteration. Once the two groups become stable, the method compares the original cluster and the two identified groups using the Bayesian information criterion in order to determine whether the cluster should be split or not. The last maintenance method is responsible for removing obsolete clusters. An event is reported in the media only for a limited number of days. To avoid assigning new articles to obsolete clusters, the method removes (micro-) clusters once the oldest member articles reach a certain age. In case of Event Registry, clusters are removed after they become 5 days old.

Both described approaches for identifying events using clustering have their advantages and disadvantages. The approach used in European media monitor works in batch mode which makes the identified events more stable. The downside is however that it is not suitable for real-time monitoring and detection of breaking events. On the other hand, the approach used by Event Registry can identify new events as soon as the sufficient number of articles about it has been written. However, because of the online mode of the algorithm, the identified clusters can be merged or split during their lifetime which makes them more volatile.

Cross Lingual Event Detection

Until this point we have not considered the fact that news articles are written in different languages. Since the described clustering approaches rely on the article text, the methods are evidently language dependent. It is not sensible, for example, to compute cosine similarity between an English and German article; therefore content from each language has to be clustered separately. As a result, events represented by the clusters will contain only articles in a single language. Since most events are reported in multiple languages, we want to find methods for identifying clusters in different languages that describe the same event. This will allow us to see how the same news is reported in different languages, what topics are more or less likely to break the language barrier, how fast does the information spreads through the languages, etc.

In order to link the appropriate clusters, we can represent the problem as a binary classification problem. Given a cluster pair c_1 and c_2 in languages l_1 and l_2, we need to compute a set of discriminative features that will help us to determine if both clusters describe the same event or not. A machine learning model can then be trained to classify the cluster pairs based on the values of the computed features.

One set of learning features can be computed by inspecting individual articles assigned to the clusters. Using a method such as canonical correlation analysis (CCA), it is possible to compute an estimated score of relatedness of two articles in different languages. The method is trained on a comparable corpus, which is a collection of documents in multiple languages, with alignment between documents that are on the same topic or even rough translations of each other. An example of such corpus is Wikipedia, where each entry can be described in multiple languages. Using the CCA, we can compare pairs of documents in the tested clusters c_1 and c_2 and compute features such as the maximum or the average score of similarity between the documents in the two clusters.

Additional set of important learning features can be computed by aggregating the annotated entities mentioned in the articles. Given the articles in each cluster, we can analyze how often do individual entities appear in the articles in order to estimate their relevance for the event – entities that appear more frequently can be considered as more important to the event compared to entities that are mentioned fewer times. One way to score an entity in a cluster is simply to compute the ratio of articles in the cluster that mention it. A more advance approach can also take into account the number of times the entity is mentioned in each article and its mentioned location – an entity is likely more relevant if it is mentioned at the beginning of the article than if at the end. Since entities are language independent (same entity, although mentioned in different languages, is represented with the same identifier), we can construct for each cluster a weighted vector of relevant entities. For a pair of clusters, a similarity measure can again be used to compute similarity of the clusters according to the mentioned entities. Since events are mostly centered around entities, the similarity score can be an important feature when deciding if two clusters are about the same event or not.

Additionally, time similarity is also an important feature. If articles in one cluster were published in a similar time period as articles in another cluster, they are more likely about the same event as if they were published several days apart. If dates mentioned in the articles are being extracted, the ratio of common dates mentioned in the two clusters can also be a relevant feature.

In order to train the classification model, we first need the learning data. A human expert should therefore provide a set of positive and negative examples – cluster pairs that are about the same events as well as pairs that are not. For each cluster pair, values of the mentioned features can be computed and concatenated into a single feature vector. A machine learning classifier, such as SVM, can then be trained to best distinguish between the positive and negative examples based on the learning features. An experiment using the described approach (Rupnik et al. 2016) reports the cluster linking accuracy of 0.893 as measured using F1 score.

Extraction of Event Properties

Based on the described approach, an event consists of one or more clusters, where each cluster contains articles from a single language. As the final step, we wish to extract from the articles in the clusters as much structured information as possible about the event.

To determine the date of the event, we can analyze the publishing date of the articles in the clusters. The simplest method can be to use the date of the first article as the date of the event. This approach can generate erroneous results for events that are reported in advance (such as various meetings of politicians, product announcements, etc.) as well as when the collected publishing dates of the articles are potentially inaccurate. A more error-prone approach is to analyze the density of reporting and use the time point where the reporting intensified as the date of the event. Additional input can be provided by the mentioned date references – a particular date that is consistently mentioned across the articles is likely the correct date of the event.

In order to determine who is involved in the event, we can analyze and aggregate the entities mentioned in the articles. A list of relevant entities and their score of relevance can be obtained by analyzing the frequency of their occurrence in the articles as well as the locations of the mentions in text. Entities that appear in event's articles more frequently and early in the text are more important than entities that are just rarely mentioned and appear late. Entities can be scored and ranked according to this criterion which provides an accurate aggregated view on what and who is the event about.

Another core property of the event is also the location where the event occurred. Since the event location is commonly mentioned in the articles, we can identify it by analyzing the frequently mentioned entities that are of type location – knowledge about the entity type can be retrieved from the knowledge base used in entity linking. Additional signal for determining the location can be obtained by inspecting the datelines of the articles. A dateline is a brief piece of text at the beginning of the news article that describes where and when the described story happened. The problem with datelines is that they are not present in all news articles, and even when they are, they sometimes represent the location where the story was written and not the actual location of the event. To determine the event location, one can simply use the city that is mentioned the most in the articles. A more advanced approach can again rely on machine learning. Each city that is mentioned in the articles about an event can be considered as a candidate for the event location. For each city we therefore generate a set of features based on which a classification model can compute the probability that it is the location of the event. The features can be the number or ratio of times the city is mentioned in the articles, the number of times it is mentioned in the dateline, how commonly the city is mentioned in all the articles, etc. To train the classification model, we again need the experts to manually provide information about the correct location of various events. Using the training data, we can then train a model that will classify each candidate city independently. Because they are evaluated independently, it is possible that the model finds several locations to be the event location. To choose the most likely city, it is important to use a probabilistic classifier that can also return a degree of certainty – in such cases, one can simply choose the location with the highest probability.

There are many other properties that could be extracted which are specific for individual event types. In case of an earthquake, for example, important properties would include the number of casualties and the strength of the earthquake. Similarly, for a football game, the relevant information would be the names of the teams that played and the final score. Identifying such properties and their values is a cumbersome task. It first requires that each event is classified into an event type (such as earthquake, football game, meeting, etc.). To perform classification, a taxonomy of event types is first needed together with a model that can perform classification into the taxonomy. Next, for each event type, a set of properties/slots need to be identified that are relevant for the event type. A pattern or rule-based approach can then be used to determine the values for these properties.

Cross-References

- ▶ Classification
- ▶ Clustering
- ▶ Cross-Lingual Text Mining
- ▶ Entity Resolution
- ▶ Text Mining

Recommended Reading

Gao J, Leetaru KH, Hu J, Cioffi-Revilla C, Schrodt P (2013) Massive media event data analysis to assess world-wide political conflict and instability. In: Social computing, behavioral-cultural modeling and prediction. Springer, Berlin/New York, pp 284–292

Leban G, Fortuna B, Brank J, Grobelnik M (2014a) Event registry: learning about world events from news. In: Proceedings of the companion publication of the 23rd international conference on World wide web companion, Seoul, pp 107–110

Leban G, Fortuna B, Brank J, Grobelnik M (2014b) Cross-lingual detection of world events from news articles. In: Proceedings of the 13th international semantic web conference, Trentino

Leetaru K, Schrodt PA (2013) GDELT: global data on events, location, and tone, 1979–2012. ISA Annu Conv 2:4

Milne D, Witten IH (2013) An open-source toolkit for mining Wikipedia. Artif Intell 194:222–239

Pouliquen B, Steinberger R, Deguernel O (2008) Story tracking: linking similar news over time and across languages. In: Proceedings of the workshop on multi-source multilingual information extraction and summarization, Manchester, pp 49–56

Rupnik J, Muhic A, Leban G, Škraba P, Fortuna B, Grobelnik M (2016) News Across Languages - Cross-Lingual Document Similarity and Event Tracking. J. Artif. Intell. Res., Special Track on Cross-language Algorithms and Applications 55, 283–316

Steinberger R, Pouliquen B, Ignat C (2005) Navigating multilingual news collections using automatically extracted information in: Proc. of the 27th International Conference on Information Technology Interfaces, pp. 25–32

Evolution of Agent Behaviors

▶ Evolutionary Robotics

Evolution of Robot Control

▶ Evolutionary Robotics

Evolutionary Algorithms

Synonyms

Evolutionary computation; Evolutionary computing; Genetic and evolutionary algorithms

Definition

Generic term subsuming all machine learning and optimization methods inspired by neo-Darwinian evolution theory.

Cross-References

- ▶ Coevolutionary Learning
- ▶ Compositional Coevolution
- ▶ Evolutionary Clustering
- ▶ Evolutionary Computation in Economics
- ▶ Evolutionary Computation in Finance
- ▶ Evolutionary Computational Techniques in Marketing
- ▶ Evolutionary Feature Selection and Construction
- ▶ Evolutionary Fuzzy Systems
- ▶ Evolutionary Games
- ▶ Evolutionary Kernel Learning
- ▶ Evolutionary Robotics
- ▶ Neuroevolution
- ▶ Nonstandard Criteria in Evolutionary Learning
- ▶ Test-Based Coevolution

Evolutionary Clustering

David Corne[1], Julia Handl[2], and
Joshua Knowles[2]
[1]Herriot-Watt University, Edinburgh, UK
[2]University of Manchester, Manchester, UK

Synonyms

Cluster optimization; Evolutionary grouping; Genetic clustering; Genetic grouping

Definition

Evolutionary clustering refers to the application of evolutionary algorithms (also known as genetic algorithms) to data clustering (or cluster analysis), a general class of problems in machine learning with numerous applications throughout science and industry. Different definitions of data clustering exist, but it generally concerns the identification of homogeneous groups of data (clusters) within a given data set. That is, data items that are similar to each other should be grouped together in the same cluster or group, while (usually) dissimilar items should be placed in separate clusters. The output of any clustering method is therefore a specific collection of clusters. If we have an objective way to evaluate (calculate the quality of) a given grouping into clusters, then we can consider the clustering task as an optimization problem. In general, this optimization problem is NP hard, and it is common to address it with advanced heuristic or meta-heuristic methods. Evolutionary algorithms are prominent among such methods and have led to a variety of promising and successful techniques for cluster optimization.

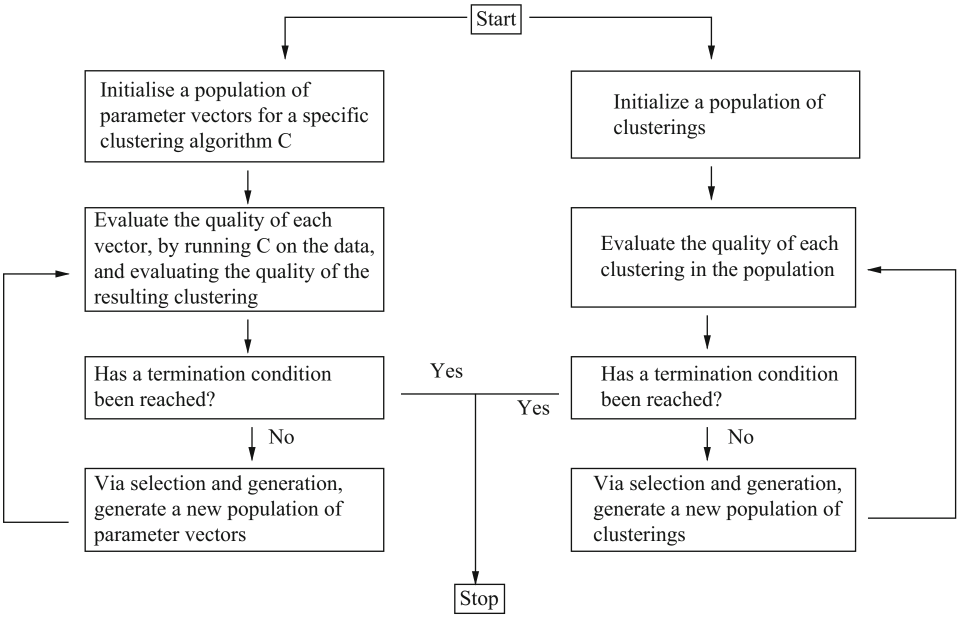
Motivation and Background

In many problem-solving scenarios, we have large amounts of data. We need to cluster those data sensibly into groups in order to help us understand the problem and decide how to proceed further (see clustering). It is common, in fact, for this initial "cluster analysis" stage to be the most important (or only) stage in the investigation. In bioinformatics, for example, a frequent activity is the clustering of gene expression data (data that indicate, for a specific cell, how active each of several thousands of genes are at different points in time or under different experimental conditions). A very important current challenge is to understand the role of each gene; by clustering such data, which means arranging genes into groups such that genes in the same group have similar patterns of activity, we find important clues about genes whose role is currently unknown, simply by assigning their putative role as being related to that of genes (whose role is known) that are in the same cluster. Meanwhile, a ubiquitous situation in industry and commerce is the clustering of data about customers or clients. Here, the role of clustering is all about identifying what types of clients (e.g., based on age, income, postcode, and many other attributes that may make up a customer's profile) buy or use certain kinds of products and services. Effective ways to identify groups enable companies to better target their products and their direct marketing campaigns and/or make more effective decisions about loans, credit, and overdrafts. Many machine learning techniques can be used to predict things about customers, predict things about genes, and so forth. However, the value of clustering (in a similar way to visualization of the data) is that it can lead to a much deeper understanding of the data, which in turn informs the continuing process of applying machine learning methods to it. In this general context, there are many well-known and well-used clustering methods, such as k-means, hierarchical agglomerative clustering, neighbor joining, and so forth. However, there are also well-known difficulties with these methods; in particular, there is often a need to choose in advance the number of clusters to find in the data, and they tend to be strongly biased toward finding certain types of groupings. For these reasons, methods that are more flexible have been recently investigated, and evolutionary clustering techniques are prominent among these. They are flexible in that (e.g., unlike k-means) the choice of the number of clusters does not have to be made a priori, and the method is not tied to any particular way of identifying the distance between two items of data, nor is there any a priori inductive bias concerning what counts as a good clustering. That is, in broad terms, an evolutionary clustering algorithm allows a user to flexibly make these decisions in view of the actual problem at hand; these decisions are then "plugged into" the algorithm which proceeds to search for good clusterings.

Given a data set to be clustered, the concept of evolutionary clustering covers two distinct ways in which we can address the problem of finding the best clustering. Each of these approaches is under continuing research and has proven successful under different conditions. The first approach is to use an evolutionary algorithm to search the space of candidate groupings of the data; this is the most straightforward approach and perhaps the most flexible in the sense discussed above. The second approach is to "wrap" an evolutionary algorithm around a simpler clustering algorithm (such as k-means) and either use the evolutionary algorithm to search the space of features for input to the clustering algorithm (i.e., the evolutionary algorithm is doing feature selection in this case) or to search a space of parameters, such as the number of clusters, feature weights, and/or other parameters of the clustering algorithm in use. Central in all of these approaches is a way to measure the quality of a clustering, which in turn depends on some given metric that provides a distance between any pair of data items. Although some applications come with pre-identified ways to measure distance and cluster quality, in the following we will assume the most common approach, in which distance is the Euclidean distance between the data items, and the measure of quality for a given clustering is some ratio of within-cluster and between-cluster similarities.

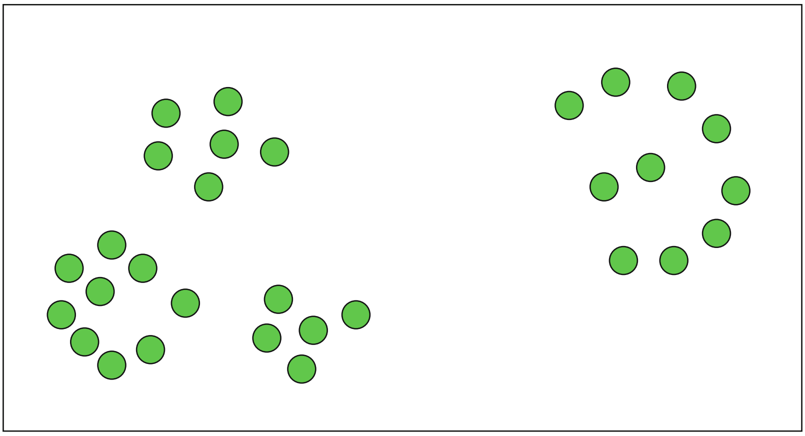
We illustrate the two main approaches to evolutionary clustering in Fig. 1.

There are several examples of the first type of approach, called "indirect" evolutionary clustering in Fig. 1 (left). This approach is often used where the "internal" clustering method ("C" in the figure) is very sensitive to initialization conditions and/or parameters of the metric in use to measure distance between items. For example, if C is the k-means algorithm, then, for each application of C, we need choices for the parameter k and for each of k initial cluster center positions in the data space. The parameter vectors referred to in the figure would be precisely these; the evolutionary algorithm searches this parameter space, finding those that lead to an optimized clustering from k-means. Figure 2 illustrates why this will often be a more effective approach than k-means alone. In this case, it is entirely unclear whether these data form two, four, or even five clusters. There are two widely separated groups of points, and this two-cluster solution may be easily found by a 2-means algorithm. However,


Start
Initialise a population of parameter vectors for a specific clustering algorithm C
Evaluate the quality of each vector, by running C on the data, and evaluating the quality of the resulting clustering
Has a termination condition been reached?
No
Via selection and generation, generate a new population of parameter vectors
Yes
Yes
Initialize a population of clusterings
Evaluate the quality of each clustering in the population
Has a termination condition been reached?
No
Via selection and generation, generate a new population of clusterings
Stop

Evolutionary Clustering, Fig. 1 Evolutionary clustering. The two main approaches to evolutionary clustering: indirect (*left*) and direct (*right*)

Evolutionary Clustering, Fig. 2 An example data set with many potential interpretations of the number of clusters



to the human eye, there is also a clear four-cluster solution, further analysis of which may lead to better understanding of these data. This four-cluster solution is difficult for a 4-means algorithm to find, depending on very fortunate initial settings for the cluster centers. The embedding of k-means within an evolutionary algorithm allows for the iterative optimization of parameters and starting conditions to arrive at this optimal solution.

On the right in Fig. 1, we see the direct approach, in which the evolutionary algorithm searches the space of clusterings of the data. The key features in this approach are the encoding and genetic operators. After evaluating the quality of each of a population of clusterings, a new population is generated from the old one via selection and variation. Essentially, some individuals from the current population are treated as “parents,” and new ones are produced from these by using genetic operators. The encoding dictates precisely how a specific data clustering is represented, while the operators determine how new clusterings are derived from the old ones. To take a simple example, suppose we needed to cluster ten items (A, B, C,. . ., J) into an arbitrary number of groups. In a simple encoding, we might represent a clustering as a vector of ten labels, independently chosen from 1 to 10, in which the *i*th element gives the group label of the *i*th item. Hence, the following individual in our population of clusterings 2 3 5 5 1 5 7 3 2 7 would represent the following grouping: (A, I) (B, H) (C, D, F) (E) (G, I). Given such a representation, a typical genetic operator might be to randomly change a single label in a single parent. For example, we may choose the fifth element in the above vector and change it randomly to 7, effectively placing item E in the same group as items G and I. Further notes about operators for this and other encodings are given in a special subsection below. Going back to the example in Fig. 2, meanwhile, it is worth noting that there are potentially five clusters, as the group on the right can be perceived as a central group of two items, surrounded by a single backward-C-shaped group. The "backward-C" cluster is an example that cannot be reliably detected (as a distinct cluster from the group of two items contained within it) with most standard cluster analysis methods. Traditional approaches typically incorporate the assumption that clusters will be centered around a particular position, with the likelihood of a point belonging to that cluster falling monotonically with its distance from that position. One of the strengths of evolutionary clustering is that it provides the flexibility to work effectively with arbitrary definitions of what may constitute a valid cluster.

Objective Functions for Evolutionary Clustering

It can be strongly argued that the clustering problem is inherently multiobjective, yet most methods employ only a single performance criterion to optimize. In fact, there are at least three groups of criteria commonly used (but usually one at a time) in clustering (both evolutionary clustering and other methods). These are compactness, connectedness, and spatial separation. When an algorithm optimizes for compactness, the idea is that clusters should consist of highly homogeneous data items only – that is, the distance (or other measure of variation) between items in the same cluster should be small. In contrast, if we optimize the degree of connectedness, then we are increasing the extent to which neighboring data items should share the same cluster. This can deal with arbitrarily shaped clusters, but can lack robustness when there is little spatial separation between clusters. Finally, spatial separation is usually used as a criterion in combination with compactness or with a measure of the balance of cluster sizes.

In multiobjective clustering, the idea is to explicitly explore the solutions that are trade-offs between the conflicting criteria, exploiting the fact that these trade-off solutions are often the ones that most appeal as intuitively "correct" solutions to a clustering problem. Handl and Knowles (2007) introduced a multiobjective evolutionary algorithm, MOCK, which treats a clustering problem as a two-objective problem, using measures of compactness and connectedness for the two objectives. MOCK's multiobjective search process is based on the PESA-II evolutionary multiobjective optimizer (Corne et al. 2001). Following the use of MOCK for a clustering problem, an intermediate result (inherent in multiobjective optimization methods) is a (possibly large) collection of different clusterings. These will range from clusterings that score very well on compactness but poorly on connectedness through clusterings that achieve excellent connectedness at the expense of poor compactness. It is useful to note that the number of clusters tends to increase as we go from poor connectedness to high-connectedness clusters. Arguably, in many applications, such a collection of alternative solutions is useful for the decision-maker. Nevertheless, the MOCK approach incorporates an automated model selection process that attempts to choose an ideal clustering from the discovered approximate Pareto front. This process is oriented around the notion of determining the "right" number of clusters and makes use of Tibshirani et al. (2001) gap statistic (full details are in Handl and Knowles 2007). Extensive comparison studies, using a wide variety of clustering problems and comparing with many alternative clustering methods, show consistent performance advantages for the MOCK's approach. Recent work has explored different objectives, encodings, and model selection mechanisms for multiobjective clustering, including the interpretation of the approximation set as a clustering ensemble (Handl and Knowles 2013).

Encodings and Operators for Evolutionary Clustering

The encoding methods used in indirect approaches to evolutionary clustering are fairly straightforward, as they only require the specification of the parameters (and, potentially, initialization points) for the clustering method(s) used. Arguably, the development of direct approaches to evolutionary clustering is more involved, as the choice of a suitable encoding method is nontrivial and has been shown to have significant impact on optimization performance.

Encodings range from the straightforward representation noted above (with the ith gene coding for the cluster membership of the ith data item) to more complex representations, such as matrix-based or permutation-based representations. Before providing a brief description of other encodings, it is worth briefly examining a well-known disadvantage of the simple encoding. Given that they have a population, evolutionary algorithms offer the opportunity to use multi-parent genetic operators – that is, we can design operators that produce a new candidate clustering given two or more "parent" clusterings. Such operators are neither mandatory nor necessarily beneficial in evolutionary algorithms, and there is much literature discussing their merits and how this depends on the problem at hand. However, they are often found helpful, especially in cases where we can see some intuitive merit in combining different aspects of parent solutions, resulting in a new solution that seems to have a chance at being good, but which we would have been immensely unlikely to obtain from single-parent operators given the current population. In this context, we can see, as follows, that the opposite seems to be the case when we use standard multi-parent operators with the simple encoding. Suppose the following are both very good clusterings of ten items:

Clustering 1: 1111122222
Clustering 2: 2222211111

Clearly, a good clustering of these items places items 1–5 together, and items 6–10 together, in separate groups. It is also clear, however, that using a standard crossover operator between these two parents (e.g., producing a child by randomly choosing between clusterings for each item in turn) will lead to a clustering that mixes items from these two groups, perhaps even combining them all into one group. The main point is that a crossover operation destroys the very relationships between the items that underpinned the fitness of the parents. One of the more prominent and influential representations for clustering, incorporating a design for far more effective multi-parent operators, was that of Falkenauer's "Grouping Genetic Algorithm," which also provides a general template for the implementation of evolutionary algorithms for grouping problems. The essential element of Falkenauer's method is that multi-parent operators recombine entire groups rather than item labels. For example, suppose we encode two clusterings explicitly as follows:

Clustering 3: (A,I,B,H)(C,G)(D,E,F,J)
Clustering 4: (A,I,B,H)(C,D,J)(E,F,G)

A Falkenauer-style crossover operator works as follows. First, we randomly choose some entire groups from the first parent and some entire groups from the second parent; the child in this case might then be:

(A,I,B,H)(C,G)(E,F,G)

in which the groups that come from the first parent are underlined. Typically, we will now have some repeated items; we remove the entire groups that contain these items and came from the first parent, in this case leaving us with:

$$\underline{(A,I,B,H)}(E,F,G)$$

The final step is to add back the missing items, placing them one by one into one of the existing groups or perhaps forming one or more new groups. The application in hand will often suggest heuristics to use for this step. In clustering, for example, we could make use of the mean Euclidean distance from items in the groups so far. Whatever the end result in this case, note that the fact that A, I, B, and H were grouped together in both parents will be preserved in the child. Similarly, the E, F, G grouping is inherited directly from a parent.

A more recent and effective approach to encoding a clustering is one first proposed in Park and Song (1998) called a link-based encoding. In this approach, the encoding is simply a list of item indices and is interpreted as follows. If the ith element in the permutation is j, then items i and j are in the same group. So, for example,

B C E E A E G C B G

represents the following grouping:

(A,B,C,D,E,H,I)(F,G,J)

Standard crossover operators may be used with this encoding, causing (intuitively) a reasonable degree of exploration of the space of possible clusterings, yet preserving much of the essential "same-group" relationships between items that were present in the parents. In Handl and Knowles (2007) it is shown why this encoding is effective compared with some alternatives. We also briefly note other encodings that have been prominent in the history of this subfield. An early approach was that of Jones and Beltramo, who introduced a "permutation with separators" encoding. In this approach, a clustering is encoded by a permutation of the items to be clustered, with a number of separators indicating cluster boundaries. For example, if we have ten items to cluster (A–J) and use S as the separator, the following is a candidate clustering:

A I B H S C G S D E F J

representing the same grouping as that of "Clustering 3" above. Jones and Beltramo offered a variant of this encoding that is a cross between the direct and indirect approaches. In their greedy permutation encoding, a clustering is represented by a permutation (with no separator characters), with the following interpretation: the first k items in the permutation become the centers of the first k clusters. The remaining items, in the order they appear, are added to whichever cluster is best for that item according to the objective function (clustering quality metric) in use.

Applications for Evolutionary Clustering

Recent work on evolutionary clustering has focused on applications of evolu- tionary clustering to data-mining problems in a variety of disciplines, including market segmentation (by Ying, Sudha, Lusch, and Brusco) and social network analysis (by Pizutti). As mentioned above, evolutionary clustering brings key advantages in terms of its accuracy, but, possibly, its most important benefit lies in the flexibility of the approach. The capability to consider and explore trade-offs with respect to multiple clustering objectives opens up new opportu- nities for data integration, particularly in the context of exploratory analytics in applications that involve diverse, noisy (and sometimes poorly understood) data sources.

Cross-References

- ▶ Clustering
- ▶ Feature Selection
- ▶ Semi-supervised Learning
- ▶ Supervised Learning
- ▶ Unsupervised Learning

Recommended Reading

Cole RM (1998) Clustering with genetic algorithms. Masters dissertation, Department of Computer Science, University of Western Australia

Corne DW, Jerram NR, Knowles JD, Oates MJ (2001) PESA-II: region-based selection in evolutionary multiobjective optimization. In: Proceedings of the GECCO, pp 283–290

Delattre M, Hansen P (1980) Bicriterion cluster analysis. IEEE Trans Pattern Anal Mach Intell 2(4):277–291

Falkenauer E (1998) Genetic algorithms and grouping problems. Wiley, New York

Handl J, Knowles J (2005) Exploiting the trade-off – the benefits of multiple objectives in data clustering. In: Evolutionary multi-criterion optimization. Springer, Berlin/Heidelberg, pp 547–560

Handl J, Knowles J (2007) An evolutionary approach to multiobjective clustering. IEEE Trans Evol Comput 11(1):56–76

Handl J, Knowles J (2013) Evidence accumulation in multiobjective data clustering. In: Evolutionary multi-criterion optimization. Springer, Berlin/Heidelberg, pp 543–557

Jain AK, Murty MN, Flynn PJ (1999) Data clustering: a review. ACM Comput Surv 31(3): 264–323

Jones DR, Beltramo MA (1991) Solving partitioning problems with genetic algorithms. In: Belew RK, Booker LB (eds) Proceedings of the fourth international conference on genetic algorithms. Morgan Kaufmann, pp 442–449

Liu Y, Ram S, Lusch RF, Brusco M (2010) Multicriterion market segmentation: a new model, implementation and evaluation. Mark Sci 29(5): 880–894

Park Y-J, Song M-S (1998) A genetic algorithm for clustering problems. In: Proceedings of the third annual conference on genetic programming. Morgan Kaufman, pp 568–575

Pizzuti C (2012) A multiobjective algorithm to find communities in complex networks. IEEE Trans Evol Comput 16(3):418–430

Tibshirani R, Walther G, Hastie T (2001) Estimating the number of clusters in a dataset via the Gap statistic. J R Stat Soc: Ser B (Stat Methodol) 63(2): 411–423

Evolutionary Computation

▶ Evolutionary Algorithms

Evolutionary Computation in Economics

Biliana Alexandrova-Kabadjova[1], Alma Lilia García-Almanza[2], and Serafín Martínez-Jaramillo[3]
[1]Banco de México, Mexico City, Mexico
[2]Directorate of Regulation and Supervision, Banco de México, Mexico City, Mexico
[3]Directorate of Financial System Risk Analysis, Banco de México, Mexico City, Mexico

Definition

Evolutionary computation (EC) is a field in computational intelligence that takes its inspiration from nature to develop methods that resolve continuous optimization and combinatorial optimization problems. When it comes to economics, it is the area of research that involves the use of EC techniques, also subclassified as evolutionary algorithms (EAs), cultural algorithms, and self-organization algorithms, among others, in order to approach topics in economic science. The algorithms, defined as generic population-based metaheuristic optimization algorithms, are developed on the basis of the concept of biological evolution and use iterative processes such as reproduction, mutation, recombination, and selection. Some of these methods, such as genetic algorithms (GAs), genetic programming (GP), evolutionary programming (EP), estimation of distribution algorithms (EDA), evolutionary strategies (ESs), memetic algorithms, harmony search, and artificial life, have been studied and applied in computer science for more than 50 years. In mainstream economics, even though we can track the early application of GAs in game theory to as long ago as 30 years, the adoption of these methods has been slow. This area of knowledge is different from the field of evolutionary economics, which does not necessarily apply EC techniques to the study of economic problems. The use of EC in economics pursues different aims. One is to overcome some of the limitations of classical economic models and to loosen some of the strong assumptions such models make.

Motivation and Background

EC techniques, among many other machine-learning techniques, have proven to be quite flexible and powerful tools in many fields and disciplines, such as computational linguistics, computational chemistry, and computational biology. Economics-affiliated fields are by no means the exception for the widespread use of these evolutionary-inspired methods. In addition to the undeniable necessity of computing in almost every aspect of our modern lives, numerous problems in economics possess an algorithmic nature. Economists should consider computational complexity to be an important analytical tool due to the fact that some of such problems belong to the class of NP-complete (The NP-complete computational complexity class is a subset of harder problems within the NP computational class, which is the set of all the decision problems which can be solved using a nondeterministic Turing machine in polynomial time (Papadimitriou 1994).) problems. This having been said, EC has been intensively used as an alternative approach to analytical methods used to tackle numerous NP-complete problems with considerable success, mainly in the areas of game theory, econometrics, and agent-based economic modeling. Game theory is a branch of applied mathematics that attempts to model an individual's strategic behavior. The first study considered to establish the fundamentals of the field is the book Theory of Games and Economic Behavior (John von and Oskar 1944). The idea behind this theory is that the success of an individual's decisions depends on the decisions of others. Whereas originally, the aim of the theory was to study competition, in which one agent does better at another expense (zero-sum games), now it has been extended to the study of a wider class of interactions among individuals. Furthermore, it is used extensively in economics, biology, and political science, among other disciplines.

The first work in economics (The first such work approached a classic game known as the prisoner's dilemma.) that involved the use of EC dates to the 1980s. In Robert and Hamilton (1981) and Robert (1987) the authors used GAs to derive strategies for the Iterated Prisoner's Dilemma (IPD). From then on, EC techniques in economics were used in areas such as macroeconomics, econometrics, game theory, auctions, learning, and agent-based models. There is even a school of thought in economics known as evolutionary economics (See, for example, Ulrich (2008) for an introduction.) in which the approach to the study of economics involves concepts in evolution but does not necessarily rely on EC techniques.

Econometrics is a field within the wider area of economics which involves the use of statistics and its tools to measure relationships postulated by economic theory (William 2003). In particular, it is applied to macroeconomic analysis to make out the relationships between the aggregated variables that explain broad sectors of an economy. One of the first applications of GP to econometrics was made by the creator of GP himself in John (1992).

Regarding agent-based computational economics, this field can be thought of as a branch of a wider-area, agent-based modeling (Wooldridge 2002). The field of agent-based modeling is not restricted to economics. It has been applied to social sciences in general (Robert 2003), to some classical and not so classical problems in computer science, and in some other disciplines. Axelrod provides an account of his experience using agent-based methodology for several problems, and he suggests that agent-based modeling can be seen as a bridge between disciplines. Axelrod and Tesfatsion provide a good guide to the literature relevant to agent-based modeling in Robert and Leigh (2006). In Shu-Heng (2007) there is a thorough introduction to agents in economics and finance. In this work, Chen conceives of the agents not just as economic agents but also as computational intelligent units.

Structure of the Evolutionary Computation in Economics

The main areas addressed by EC in economics are game theory, econometrics and economic models, and agent-based economic modeling. In game theory, a well-defined mathematical object, the game, consists of a set of players and a set of strategies (decisions) available to those players. In addition, for each combination of strategies, specification of payoffs is provided. The aim of traditional applications of game theory was to find a Nash equilibrium, a solution concept, in which each player of the game adopts a strategy that is unlikely to be changed. This solution concept was named after John Nash, whose work was published in the early 1950s (John 1950). Nevertheless, it took almost 20 years to fully realize what a powerful tool Nash had created. Nowadays, game theory is one of the best established theories in economics, and it has been used extensively to model interactions among economic agents. However, games typically have many Nash equilibria, and one key assumption is that the agents behave in a rational way. In more realistic games, the equilibrium selection problem does not have an easy solution. Human behavior observed in real life, indeed, is frequently irrational. Given these constraints, evolutionary game theory was proposed as an application of the mathematical theory of games to biological contexts. In this field, Maynard Smith is considered to be the first to define the concept of an evolutionary stable strategy in John Maynard (1972). Furthermore, the possibility of using computer modeling as an extension of game theory was first explored in Robert and Hamilton (1981). Since then, computer science has been used in traditional game theory problems, like the strategic behavior of agents in auctions, auction mechanism design, etc. By providing approximate solutions to such complex problems, this approach can be useful where analytical solutions have not been found. For instance, the iterative prisoner's dilemma is one of the games most studied by researchers from computer science (Robert 1987). The prisoner's dilemma is a classic game that consists of the decision-making process for two prisoners who can choose to cooperate or defect from a group. In the case that the two prisoners choose to cooperate, they get a payoff of three each. In the case that both choose to defect, they get a payoff of one each, and in the case that one decides to defect and the other to cooperate, the former gets a payoff of five and the later a payoff of zero. In equilibrium, both players decide to defect despite the fact that it would be better for them to cooperate.

Game theory is one of the most important areas of economics because it has applications to many other fields, such as corporate decision-making, microeconomics, market modeling, public policy analysis, and environmental systems. We can find more applications of EC to game theory than IPD. For example, other work related to game theory and EC is that done by John and Engle-Warnick (2001), which deals with the well-known two-player, repeated ultimatum game. In this work they used GP as a means of inferring the strategies that were played by subjects in economic decision-making experiments. Other research related to game theory includes the duopoly and oligopoly games (Shu-Heng and Ni 2000). References regarding cooperation, coalition, and coordination are also frequent and usually driven by EC techniques (Vriend 1995). In 2006, the authors applied GP to find strategies for sequential bargaining procedure and confirmed that equilibria can be approximated by GP. This provides an opportunity to find approximate solutions to more complex situations for which theoretical solutions have yet to be found.

Regarding econometrics, in Adriana and Alexandr (2001) the authors use GAs and simulated annealing (SA) for econometric modeling; they found that the performance of the evolutionary algorithms (EAs) is better than the performance of traditional gradient techniques on the specific models in which they performed the comparison. Finally, Ralf Östermark (1999) uses a hybrid GA in several ill-conditioned econometric and mathematical optimization problems with good results.

In addition to the use of EC in econometrics, some classical economic models like the cobweb model and exchange-rate models have been approached with EC techniques. For instance, in Jasmina (1994) and Shu-Heng and Chia-Hsuan (1996), in the former work, the author uses GAs to approach the cobweb model, whereas in the latter the authors use GP. Furthermore, Arifovic explores the use of GAs in foreign exchange markets in Jasmina (1996). The GA mechanism elaborated in such works developed decision rules that were used to determine the composition of agents' portfolios in a foreign exchange market. Arifovic made two observations rarely seen in the standard overlapping generations (OLG) model with two currencies. First, she noted that the returns and exchange rates were generated endogenously and, second, that the models' equilibrium dynamics were not stable and showed bounded oscillations (the theoretical model implies a constant exchange rate).

The use of GAs in economic modeling is not restricted to the abovementioned works. In James et al. (1995), the authors studied a version of the growth model in which physical capital is accumulated in a standard form, but human capital accumulation is subject to increasing returns. In their model, the agents make two decisions when they are young: how much to save by renting physical capital to the companies and how much to invest in training. Returns on training depend on the average level of human capital in the economy. The authors introduce agents' learning by means of GAs. In 1990, Marimon develops an economic model in which the agents adapt by means of a GA.

The final approach is agent-based computational models built with EAs for applications in economics (ACEs). In ACEs, one of the main goals is to explain the macro-dynamics of an economy by means of the micro-interactions of the economic agents. This approach to the study of an economy has been called a bottom-up approach in contrast to more traditional approaches. An additional purpose of ACEs is to handle real-world issues, something now possible due to technological advances in computational tools.

Nevertheless, to achieve a realistic representation of the agent in a model allows us to start with a critical revision of the assumptions behind classical economic theory. One of the most important concepts in this context is rationality. It is at the core of most economic models. It is frequently assumed that economic agents behave in a fully rational way. Unfortunately, it is not clear what this assumption holds, especially in view of irrational behavior observed during recurrent financial crises.

Herbert A. Simon is probably the best known scientist to claim that "decision-making" under uncertainty is not a fully rational process. He developed his theory based on the concept of bounded rationality (Herbert 1957). He was one of the pioneers in the field of artificial intelligence (AI), as well as a highly respected psychologist and economist. Later, in Brian (1991), the author made important contributions to the development of agents with bounded rationality using computational tools. Some more recent ideas about rationality from a computer scientist's point of view are found in Edward (2008).

Some other common assumptions behind classical economic theory are that the participants of the model have *homogeneous preferences* and they *interact globally* (Robert 2000). Departing from the assumption of full rationality and homogeneous expectations, the horizon and the design issues vary widely. The modeling of the learning behavior of the agents is a central part of the research agenda in computational economics. Regarding the agents' learning process, Lucas' definition for adaptive behavior from the economic point of view is of extreme importance (Robert 1986). There are many useful techniques to implement this adaptive learning. The application of genetic algorithms

(GAs) in James and John (1999) and genetic programming (GP) in Serafin and Edward (2009) are good examples. GP has been previously described as a suitable way to model economic learning in Bruce (1999). In Thomas (2006), the author provides us a summary of the available options to model agent behavior and learning in economics.

With the use of programming languages, the agent-based approach allows us to represent explicitly agents with bounded rationality and heterogeneous preferences. Given a specific social structure, the simulation of the interaction among agents is the strength and heart of agent-based modeling (ABM). Nowadays ABM is a promising area of research, which has opened the way to social scientists to look for new insights in resolving important real-world issues. Considered the third way of doing science (Robert 2003), modeling the behavior of the autonomous decision-making entities allows researchers to simulate the emergence of certain phenomena in order to gain better understanding of the object of study (Robert 2000). In this sense ACE, defined as the computational study of economic processes modeled as dynamic systems of interacting agents (Leigh 2006), is a growing area inside the field of agent-based modeling. ACE research is developing rapidly. By using machine-learning techniques, researchers model the agents as software programs able to make autonomous decisions. Consequently, the interactions among the individuals at the microlevel give rise to regularities at the macrolevel (globally). The intention is to observe the emerging self-organizing process for a certain period of time, in order to study the presence of patterns or the lack of them. Currently, the study of this self-organizing capability is one of the most active areas of ACE research. EAs have been used for the modeling of the agents' learning in multi-agent simulations. In economics, it is possible to find very different approaches and topics. The following is a small selection from a large body of literature:

Electricity Markets (Massoud 2002) (Learning Classifier System)

Foreign Exchange Markets (Jasmina 1994; Kiyoshi and Kazuhiro 2001) Genetic Algorithms

Payment Card Markets (Biliana et al. 2011) (Population Based Incremental Learning)

Retail Petrol Markets (Heppenstall et al. 2007) (Genetic Algorithms)

Stock Markets (Brian et al. 1997) (Learning Classifier Systems) and;

(Serafin and Edward 2009) (GP)

Cross-References

- ▸ Evolutionary Algorithms
- ▸ Evolutionary Computation in Finance
- ▸ Evolutionary Computational Techniques in Marketing
- ▸ Genetic and Evolutionary Algorithms
- ▸ Genetic Programming

Recommended Reading

Agapie A, Agapie A (2001) Evolutionary computation for econometric modeling. Adv Model Optim 3(1): 1–5

Alexandrova-Kabadjova B, Tsang E, Krause A (2011) Competition is bad for consumers: analysis of an artificial payment card market. J Adv Comput Intell Intell Inform 15:188–196

Amin M (2002) Restructuring the electric enterprise: simulating the evolution of the electric power industry with intelligent adaptive agents. In: Faruqui A, Eakin K (eds) Market analysis and resource management, chapter 3. Kluwer Academic Publishers, Boston/Dordetch/London

Arifovic J (1994) Genetic algorithm learning and the cobweb model. J Econ Dyn Control 18:3–28

Arifovic J (1996) The behavior of the exchange rate in the genetic algorithm and experimental economics. J Political Econ 104:510–541

Arthur WB (1991) Learning and adaptive economic behavior. Designing economic agents that act like human agents: a behavioral approach to bounded rationality. Am Econ Rev 81:353–359

Arthur WB, Holland JH, LeBaron B, Palmer RG, Talyer P (1997) Asset pricing under endogenous expectations in an artificial stock market. In: Brian Arthur W, Durlauf S, Lane D (eds) The economy as an evolving complex system II. Addison-Wesley, Reading

Axelrod R (1987) The evolution of strategies in the iterated prisoner's dilemma. Genetic algorithms and simulated annealing of research notes in AI, chapter 3. Pitman/Morgan Kaufmann, London/Los Altos, pp 32–41

Axelrod R (2003) Advancing the art of simulation in the social sciences. Jpn J Manag Inf Syst, Spec Issue Agent-Based Model 12(3):16–22

Axelrod R, Hamilton WD (1981) The evolution of cooperation. Science 211:1390–1396

Axelrod R, Tesfatsion L (2006) A guide for newcomers to agent-based modeling in the social sciences. In: Judd KL, Tesfatsion L (eds) Handbook of computational economics, volume 2: agent-based computational economics. Handbooks in economics, chapter Appendix A. North-Holland Amsterdam, pp 1647–1656

Axtell R (2000) Why agents? On the varied motivations for agent computing in the social sciences. Working paper 17, Center on Social and Economic Dynamics

Brenner T (2006) Agent learning representation advice in modelling economic learning. In: Judd KL, Tesfatsion L (eds) Handbook of computational economics, volume 2: agent-based computational economics. Handbooks in economics, chapter 18. North-Holland, pp 895–948

Bullard J, Arifovic J, Duffy J (1995) Learning in a model of economic growth and development. Working paper 1995-017A, Federal Reserve Bank Of St. Louis

Bullard J, Duffy J (1999) Using genetic algorithms to model the evolution of heterogeneous beliefs. Comput Econ 13:41–60

Chen S-H (2007) Editorial: computationally intelligent agents in economics and finance. Inf Sci 177(5):1153–1168

Chen S-H, Ni CC (2000) Simulating the ecology of oligopolistic competition with genetic algorithms. Knowl Inf Syst 2(2):285–309

Chen S-H, Yeh C-H (1996) Genetic programming learning in the cobweb model with speculators. In: International computer symposium (ICS'96). Proceedings of international conference on artificial intelligence. National Sun Yat-Sen University, Kaohsiung, R.O.C., 19–21, pp 39–46

Duffy J, Engle-Warnick J (2001) Using symbolic regression to infer strategies from experimental data. In: Chen S-H (ed) Evolutionary computation in economics and finance. Physica-Verlag, New York, pp 61–82

Edmonds B (1999) Modelling bounded rationality in agent-based simulations using the evolution of mental models. In: Brenner T (ed) Computational techniques for modelling learning in economics. Kluwer, Boston, pp 305–332

Greene WH (2003) Econometric analysis, 5th edn. Prentice Hall, Upper Saddle River, 07456

Heppenstall A, Evans A, Birkin M (2006) Using hybrid agent-based systems to model spatially-influenced retail markets. J Artif Soc Soc Simul 9(3): 2

Izumi K, Ueda K (2001) Phase transition in a foreign exchange market-analysis based on an artificial market approach. IEEE Trans Evol Comput 5(5):456–470

Jin N, Tsang EPK (2006) Co-adaptive strategies for sequential bargaining problems with discount factors and outside options. In: Proceedings of the IEEE congress on evolutionary computation, Vancouver. IEEE Press, pp 7913–7920

Koza J (1992) A genetic approach to econometric modelling. In: Bourgine P, Walliser B (eds) Economics and cognitive science. Pergamon Press, Oxford/New York, pp 57–75

Lucas RE (1986) Adaptive behavior and economic theory. In: Hogarth RM, Reder MW (eds) Rational choice: the contrast between economics and psychology. University of Chicago Press, Chicago/London, pp 217–242

Marimon R, McGrattan E, Sargent TJ (1990) Money as a medium of exchange in an economy with artificially intelligent agents. J Econ Dyn Control 14:329–373

Martinez-Jaramillo S, Tsang EPK (2009) An heterogeneous, endogenous and coevolutionary gp-based financial market. IEEE Trans Evol Comput 13:33–55

Nash J (1950) The barganing problem. Econometrica 18:155–162

Östermark R (1999) Solving irregular econometric and mathematical optimization problems with a genetic hybrid algorithm. Comput Econ 13(2): 103–115

Papadimitriou C (1994) Computational complexity. Addison-Wesley, Reading

Simon HA (1957) Models of man: social and rational. John Wiley and Sons, Inc., New York

Smith JM (1972) Game theory and the evolution of fighting. Edinburgh University Press, Edinburgh, pp 8–28

Tesfatsion L (2006) Agent-based computational economics: a constructive approach to economic theory. In: Judd KL, Tesfatsion L (eds) Handbook of computational economics, volume 2: agent-based computational economics. Volume 2 of handbooks in economics, chapter 16. North-Holland Amsterdam, pp 831–880

Tsang EPK (2008) Computational intelligence determines effective rationality. Int J Autom Comput 5:63–66

von Neumann J, Morgenstern O (1944) Theory of games and economic behavior. Princeton University Press, Princeton

Vriend NJ (1995) Self-organization of markets: an example of a computational approach. Comput Econ 8: 205–231

Witt U (2008) Evolutionary economics. In The New Palgrave Dictionary of Economics. Second Edition. Eds. Steven N. Durlauf and Lawrence E. Blume. Palgrave Macmillan, London

Wooldridge M (2002) An introduction to multiAgent systems. Wiley, Chichester

Evolutionary Computation in Finance

Serafín Martínez-Jaramillo[1], Tonatiuh Peña Centeno[2], Biliana Alexandrova-Kabadjova[3], and Alma Lilia García-Almanza[4]
[1]Directorate of Financial System Risk Analysis, Banco de México, Mexico City, Mexico
[2]German Center for Neurodegenerative Diseases, Banco de México, Mexico City, Mexico
[3]Banco de México, Mexico City, Mexico
[4]Directorate of Regulation and Supervision, Banco de México, Mexico City, Mexico

Definition

Evolutionary computation (EC) in finance is an area of research and knowledge which involves the use of EC techniques in order to approach topics in finance. This area of knowledge is similar to EC in economics; in fact, the areas frequently overlap in some of the topics they approach. The application of EC in finance pursues two main purposes: first, to overcome the limitations of some theoretical models, also departing from some of the assumptions made in those models, and, second, to innovate in this extremely competitive area of research, given the powerful economic incentives to do so.

EC techniques have been widely used in a variety of topics in finance. Among the most relevant we find: financial forecasting, algorithmic and automatic trading, option pricing, portfolio optimization, artificial financial markets, credit rating, credit scoring, bankruptcy prediction, and filtering techniques.

Motivation and Background

Evolutionary computation (EC) is a field in machine learning (ML) in which the techniques developed apply the principle of evolution in different ways. Among the many techniques which have been used in financial applications, one can find genetic algorithms (GAs), genetic programming (GP), learning classifier systems (LCSs), population-based incremental learning (PBIL), grammatical evolution (GE), evolutionary strategies (ESs), memetic algorithms (MAs), and evolutionary nearest neighbor classifier algorithm (ENPC), among many others. In addition, many of the above mentioned techniques are used in combination or as meta-techniques on top of other machine-learning tools. In many financial markets, competition is at the center of everyday activities undertaken by individuals and companies. As a consequence, given this fierce competition and the necessity for innovation, it is natural to find numerous problems in finance being approached by existing EC techniques. For example, in stock markets, individual and institutional investors try to beat the market in order to make more profits than other market participants. Coming up with novel algorithms or techniques is crucial to maintaining their performance and status in relation to competitors.

This area of research has been given many different names, including computational finance and computational intelligence in finance, among others. Research in this area is still evolving. Therefore, it is difficult to define the field clearly or to establish its limits. Moreover, nowadays it is almost impossible to provide a full account of all the relevant work that involves any form of EC in finance. It is also hard to organize the vast amount of human knowledge implicit in the field. The number of specialized journals, meetings, and books is indeed very large and getting larger. Chen (2002a), Chen and Wang (2004), and Chen et al. (2007) exemplify important research in this dynamic field.

Computing in finance is an almost unavoidable tool, from Monte Carlo simulation to computer intensive methods used to price complex derivatives. Furthermore, some of the most crit-

The views expressed here are those of the authors and do not represent the views of the Mexican central bank. The authors are grateful with Alberto Romero Aranda and Dorothy Walton for their valuable comments on this entry.

ical processes in finance make heavy use of computers. Computational finance is a frequently mentioned term, sometimes associated with financial engineering. However, in this context we refer to computational finance as the use of nonconventional computational techniques, like EC or other machine-learning techniques, to tackle problems in finance. See, for example, Tsang and Martinez-Jaramillo (2004) for a good introduction to the field. Additionally, Chen (2002b), Brabazon and O'Neill (2008), and Brabazon and O'Neill (2009) illustrate relevant works in the field.

Financial Forecasting and Algorithmic and Automatic Trading

In recent years, computers have shown themselves to be a powerful tool in financial applications. For that reason, many machine-learning techniques have been applied to financial problems. Financial forecasting is one of the most important fields in the area of computational finance (Tsang and Martinez-Jaramillo 2004). EC has been used to solve a great variety of financial forecasting problems, such as prediction of stock prices changes and their volatility, forecasting in foreign exchange markets, and more. Let us introduce some of the most important research in the financial forecasting area. This does not pretend to be either an extensive or detailed survey of literature in the field. The objective is just to illustrate the use of EC in financial forecasting applications.

Machine-learning classifiers, like other forecasting techniques, extend past experiences into the future. The aim is to analyze past data in order to identify patterns in the interest of creating a model or a set of rules to predict future events. In particular, EC techniques have some characteristics that make them useful for financial forecasting. For example, evolutionary techniques are able to produce interpretable solutions. This characteristic is especially important for predictions, since the main goals of classification are to (1) generate an accurate classification model that should be able to predict unseen cases and (2) discover the predictive structure of a problem (Breiman et al. 1984).

Models which help to understand the structural patterns in data provide information that can be useful for recognizing the variables' interactions. There are classification models that have good predictive power. However, these models provide a poor representation of the solution (take, e.g., the artificial neural networks). Since EC techniques provide not just good predictions but interpretable solutions, they have been used in financial problems to acquire knowledge of the event to predict. For example, Tsang et al. (2004) trained a GP using past data from the financial stock markets to predict price movements of at least $r\,\%$ within a period of at most n time units. The attributes used to train the GP were indicators from technical analysis. Due to the possibility of interpreting the solution, the authors were able to analyze the most successful indicators in the result. In fact, some researchers have used EC in order to discover new financial indicators. This include Allen and Karjalainen (1999), who made use of a GP system to infer technical trading rules from past prices. The algorithm was applied to the S&P 500. Bhattacharyya et al. (2002) used GP to discover trading decision models from high-frequency foreign exchange (FX) market data.

In other related works, Bhattacharyya et al. (2002) used GA for mining financial time series to identify patterns, with the aim of discovering trading decision models. Potvin et al. (2004) applied GP to automatically generate short-term trading rules on the stock markets. The authors used historical pricing and transaction volume data reported for 14 Canadian companies from the Toronto Stock Exchange market. Another approach called grammatical evolution (GE) (Brabazon and O'Neill 2004) was applied to discover new technical trading rules, which can be used to trade on foreign exchange markets. In that approach, each of the evolved programs represents a market trading system.

Additionally, EC techniques are able to generate a set of solutions for a single problem. This characteristic has been used to obtain a set of results with the aim of applying the most suitable solution to the particular problem. For instance, Lipinski (2004) analyzed high-frequency data. The independent variables were composed by 350 expert rules and observations of stock price quotations and order books recorded from the Paris Stock Exchange. In that model, stock market trading rules were combined into stock market trading experts, which defined the trading expertise. The author used a simple GA, a population-based incremental learning (PBIL), a compact genetic algorithm (CGA), and an extended compact genetic algorithm (ECGA) to discover optimal trading experts in a specific situation. The author argues that the optimal solution depends on the specific situation in the stock market, which varies with time. Thus, optimal trading experts must be rebuilt. EC plays an important role in learning and continual adaptation to the changing environment.

Taking advantage of the EC's ability to generate multiple solutions, Garcia-Almanza and Tsang (2008) proposed an approach, called Evolving Comprehensible Rules (ECR), to discover patterns in financial data sets to detect investment opportunities. ECR was designed to classify the minority class in unbalanced environments, which is particularly useful in financial forecasting given that very often the number of profitable opportunities is scarce. That approach offers a range of solutions to suit an investor's risk guidelines. Thus, the user can choose the best trade-off between misclassification and false alarm costs according to the investor's requirements. The approach proposed by Ghandar et al. (2008) was designed to generate trading rules. The authors implemented an adaptive computational intelligent system by using an evolutionary algorithm and a fuzzy logic rule-based representation. The data used to train the system was composed just of volume and price. The authors' objective was to create a system to generate rules to buy recommendations in dynamic market conditions. An analysis of the results was provided by applying the system for portfolio construction to historical data for companies listed on the MSCI Europe Index from 1990 to 2005. The results showed that their approach was able to generate trading rules that beat traditional fixed rule-based strategies, such as price momentum and alpha portfolios, and the approach also beat the market index.

Given that EC can be used as an optimization technique, EC techniques have been combined with other approaches. For example, Chen et al. (1999) used a genetic algorithm to determine the number of input variables and the number of hidden layers in an NN for forecasting Dollar/Deutsche mark foreign exchange rates. Chen and Lu (1999) used GP to optimize a NN. That approach is called evolutionary neural trees (ENTs). The objective was to forecast the high-frequency stock returns of the Heng Seng stock index. Schoreels et al. (2004) investigated the effectiveness of an agent-based trading system. The system employs a simple GA to optimize the trading decisions for every agent; the knowledge is based on a range of technical indicators generating trading signals. In Dempster et al. (2001) the authors aim to detect buy and sell signals in the FX markets. The authors analyze and compare the performance of a GP combined with a reinforcement learning (RL) system to a simple linear program (LP) characterizing a Markov decision process (MDP) and a heuristic in high-frequency (intraday) FX trading. The authors consider eight popular technical indicators used by intraday FX traders based on simple trend indicators such as moving averages as well as more complex rules. From experimental results, the authors found that all methods were able to create significant in-sample and out-of-sample profits when transaction costs are zero. The GP approach generated profits for nonzero transaction costs, although none of the methods produce significant profits at realistic transaction costs.

As is evident, EC techniques allow the representation of solutions using different structures, such as decision trees (Potvin et al. 2004), finite-state automata, graphs, grammar (Brabazon and O'Neill 2004), networks, and binary vectors (Lipinski 2004), among many others. This characteristic lets us choose the best representation for the problem.

Portfolio Optimization

Portfolio optimization is an all-important field in finance. The portfolio selection problem can be described in a simple way as the problem of choosing the assets and the proportion of such assets in an investor's wealth in an effort to maximize profits and minimize risk.

As the name suggests, *portfolio optimization* is an optimization problem and EC has proven to be very useful in difficult (sometimes intractable) optimization problems. In Maringer (2005), the author explains extensively the portfolio optimization problem and the possible heuristic approaches, including Ant Systems (AS), memetic algorithms (MAs), genetic algorithms (GAs), and evolutionary strategies (ESs). For an extensive review from a financial economic perspective, see Brandt (2009).

Being a multi-objective optimization problem, EC provides plenty of opportunities to approach the portfolio optimization problem. For example, Hassan and Clack (2008) uses a multi-objective GP to approach this problem. In Diosan (2005), the author compares different multi-objective evolutionary algorithms for the portfolio optimization problem.

The number of papers on portfolio optimization using machine-learning techniques is large. Streichert et al. (2004), Doerner et al. (2004), and Maringer (2006) are some significant works on portfolio optimization that use some form of evolutionary computation or artificial intelligence.

Multi-objective evolutionary optimization is an important field within EC, and the portfolio optimization problem is not the only application in finance which can be approached. In Coello (2006), the author surveys the literature on multi-objective optimization in economics and finance.

Financial Markets

This section introduces the applications of EC in artificial financial markets. Due to the extensiveness of the literature, only a general overview will be provided. For a more complete and detailed guide to the applications of EC techniques in artificial financial markets, see Martinez-Jaramillo and Tsang (2009a).

Financial markets are essential for financial systems. Such markets represent one of the most efficient ways to allocate financial resources to companies. However, bubbles and crashes are recurrent phenomena which have enormous repercussions for the global economy. Indeed, nowadays we can see as never before that one single crash in one market can lead to a worldwide slump on most of the other stock markets. Moreover, crisis in financial markets can affect other aspects of the (real) economy, for example, interest rates, inflation, unemployment, etc. This, in turn, can cause even more instability on the financial markets.

Financial markets are very important in our lives, whether we like it or not. For example, everyone suffers the consequences of a stock market crash such as the international market crash in 1987. Moreover, this phenomena (market crashes) occurs with an unpleasantly higher frequency than predicted by standard economic theory. Important references on rare disasters and asset markets are Barro (2009), Gabaix (2012), and Gourio (2008). One of the most important research issues in financial markets is an explanation for the process that determines asset prices and, as a result, rates of return. There are many models that can be used to explain such processes, such as the capital asset pricing model (CAPM) (Sharpe 1964), arbitrage pricing theory (APT) (Ross 1976), or Black-Scholes option pricing (Black and Scholes 1973).

Nevertheless, financial markets are very complex to analyze due to the wide variety of participants and their ever-changing nature. The most common approach to study them is by means of analytical models. However, such models have some limitations which, in turn, have led to the search for alternative methods to approach them. Agent-based computational economics (ACE) (Tesfatsion 2002) and computational finance (Tsang and Martinez-Jaramillo 2004) have risen as alternative ways to overcome some of the problems of the analytical models.

Agent-based financial markets with varying characteristics have been developed for the study of such markets in the last decade, since the influential Santa Fe Artificial Market (The Santa Fe Artificial Stock Market is a simulated stock market developed at the Santa Fe Institute. The market was developed by a team of highly regarded researchers, among them is John Holland, the inventor of genetic algorithms Holland 1975.) (Arthur et al. 1997). Some of them differ from the original Santa Fe market in the type of agents used, such as Chen and Yeh (2001), Gode and Sunder (1992), Yang (2002), and Martinez-Jaramillo and Tsang (2009b), and in market mechanisms, such as Bak et al. (1997), Gode and Sunder (1992), and Yang (2002). Other markets borrow ideas from statistical mechanics, such as Levy et al. (1994) and Lux (1998). Some important research has been done modeling stock markets inspired by the minority game (The minority game was first proposed by Yi-Cheng Zhang and Damien Challet (1997) inspired by the El Farol bar problem introduced by Brian Arthur 1994.) like Challet et al. (2000). There are financially simulated markets in which several stocks are traded, such as in Cincotti et al. (2005). However, criticism of this approach centers on the problem of calibration, the numerous parameters needed for the simulation program, and the complexity of simulation, among other problems. The contradictions between existing theory and the empirical properties of stock market returns are the main driving force for some researchers to develop and use different approaches to study financial markets. An additional aspect of the study of financial markets is the complexity of the analytical models of such markets. Prior to the development of some new simulation techniques, very important simplifying (unrealistic) assumptions had to be made in order to allow for the tractability of the theoretical models.

Artificial intelligence and, in particular, EC have been used in the past to study financial and economic problems. However, the development of a well-established community known as the agent-based computational economics community facilitates the study of phenomena in financial markets that was not previously possible. Within this community, a vast number of studies and approaches are being produced in order to solve or gain more understanding of economic problems.

The influential study (Arthur et al. 1997) and previously the development of the concept of bounded rationality in Simon (1982) and Arthur (1991) changed the way in which we conceive and model economic agents. This change in conception dramatically altered the possibilities for studying some economic phenomena and, in particular, financial markets. The new models of economic agents have changed. There is no longer any need for fully rational representative agents or for homogeneous expectations and information symmetry. Furthermore, the development of artificially adapted agents (Holland and Miller 1991) provides a way forward for economic science to study economic systems.

Although they all differ in the sorts of assumptions made, methodology, and tools, these markets share the same essence: the macrobehavior of such markets (usually the price) should emerge endogenously as a result of the microinteractions of the (heterogeneous) market participants. This approach is in opposition to traditional techniques used in economics and finance. Moreover, in Lux and Ausloos (2002) the authors declare:

> Unfortunately, standard modelling practices in economics have rather tried to avoid heterogeneity and interaction of agents as far as possible. Instead, one often restricted attention to the thorough theoretical analysis of the decisions of one (or few) *representative* agents.

The *representative agent* is a common, yet very strong, assumption in the modeling of financial markets. This concept has been the source of controversy and strong criticism. For example, in Kirman (1992), the author criticizes the *representative individual* approach in economics.

In order to understand the approaches in artificial (simulated) financial markets, it is useful to describe the different types of markets on the basis of the framework proposed in LeBaron (2001). In this study, LeBaron identifies the key design issues present in every artificial financial market and describes some of the most important studies up to then. In LeBaron (2006), LeBaron surveys again the literature existing until then. The main design issues identified in LeBaron (2001) are:

- Agents
- Market mechanisms
- Assets
- Learning
- Calibration
- Time

In addition to describing the different approaches in artificial financial markets by using the above-described framework, there is a fairly detailed extension of it in Grothmann (2002) that is worth looking at. In this study, the basic design issues proposed in LeBaron (2001) are extended and given more detail.

Option Pricing

Derivatives (See Hull (2008) for an introduction to derivatives.) are financial instruments whose main purpose is to hedge risk. However, derivatives can also be used to speculate with very negative effects on the financial health of companies, as we all know now. Derivative markets have seen significant expansion in recent years. Futures, forwards, swaps, and options are the best known types of derivatives. Option pricing is an extremely important task in finance. The Black-Scholes model for option pricing is the reference analytical model since it has an important theoretical framework behind it. However, in practice, prices deviate from the prices obtained with this model. One possible reason for the departure is the assumptions being made in the model (the assumption of constant volatility and the assumption that prices follow a geometric Brownian motion). This is why GP was used as an alternative to perform option pricing in Chen et al. (1998), Chidambaran et al. (2002), Fan et al. (2007), and Yin et al. (2007). Interestingly, not only has GP been used to perform option pricing, but also ant colony optimization (ACO) has been explored to approach this important problem in finance (Kumar et al. 2008).

Credit Rating, Credit Scoring, and Bankruptcy Prediction

Credit rating and credit scoring are two examples of financial problems that have been traditionally approached through statistical analysis. A credit rating is an estimate of a corporation's worthiness to be given a credit and is generally expressed in terms of an ordinal value. Credit scoring is a technique used to express the potential risk of lending money to a given consumer in terms of a probability measure. Both techniques are similar in their ends but applied to different domains.

The seminal work in the field of credit scoring is that of Altman (1968), who proposed the application of linear discriminant analysis (Fisher 1936) to a set of measurements known as financial ratios, i.e., indicators of a corporation's financial health obtained from the corporation's financial statements. One of the main applications of Altman's method, also known as the Z-score, is bankruptcy prediction. Understandably, a series of improvements have been achieved by means of applying more powerful classifiers, such as decision trees, genetic programming, neural networks, and support vector machines, among others. References that apply such techniques or conduct a review of the literature on their application are Atiya (2001), Sung et al. (1999), West (2000), Ong et al. (2005), Shin and Lee (2002), Martens et al. (2007), and Huang et al. (2007).

Another method to evaluate credit worthiness is that provided by specialized agencies. The so-called credit ratings are nothing more than ordinal values expressing the financial history, current assets, and liabilities of entities such as individuals, organizations, or even sovereign countries, such that they represent the likelihood of default on any type of debt. Although each rating agency uses its own methodology and scale and these are usually not disclosed, in the academic realm, nevertheless, several superseding techniques to ordinal regression have been applied. For example, Huang et al. (2004), Dutta and Shekhar (1988), Paleologo et al. (2009), and Zhou et al. (2006) have proposed computationally oriented methods to solve this problem.

Related to bankruptcy prediction, NNs have been the standard selection apart from the traditional statistical methods (discriminant analysis, logit and probit models). Quintana et al. (2008) explore the feasibility of using the evolutionary nearest neighbor classifier (ENPC) algorithm suggested by Fernández and Isasi (2004) in the domain of early bankruptcy prediction. They assess its performance comparing it to six alternatives; their results suggest that this algorithm might be considered as a good choice. Another relevant study is Turku et al. (1996) in which the authors compare discriminant analysis, logit analysis, and GAs for the selection of the independent variables used for the prediction model.

Filtering Techniques

Many real-life problems involve the estimation of unknown data from observed (probably noisy) values. Direct estimation methods like the Markov chain Monte Carlo (Andrieu et al. 2003), the sequential Monte Carlo (Doucet et al. 2001), and the particle filter (Gordon et al. 1993) methods are very useful for this task. In addition to the many applications of filtering techniques, filters are also very important tools in finance and economics. Their applications in the fields of macroeconomics, microeconomics, and finance are numerous. To enumerate them all is beyond the scope of this entry.

Among the many variations of filtering techniques, the Kalman filter (Kalman 1960), the extended Kalman filter (Jazwinski 1970), the unscented Kalman filter (Julier and Uhlmann 1997), the particle filter (Gordon et al. 1993), and the hidden Markov model (Baum et al. 1970) are some of those which practitioners use most widely in finance. In economics the Hodrick-Prescott filter (Hodrick and Prescott 1997) is one of the most widely used.

These methods have benefited from the application of EC techniques to optimize over the parameter space and to improve the performance of the methods in particular applications. For example, in O'Sullivan (2007), the authors optimize over the parameter space by using an evolutionary optimizer known as differential evolution (DE) for a Cox, Ingersoll, and Ross term-structure model. The authors in Rezaei et al. (2008) make use of EC techniques to improve the performance of a Kalman filter by means of GAs. In Kumar et al. (2010) the authors tune an extended Kalman filter using different EAs.

In an interesting application of EC techniques in the tuning of Kalman filters, (Huo et al. 2014) determines the initial parameterization of a Kalman filter with a GA, and the parameterization is adaptively updated by means of a Fuzzy Inference System (FIS).

Cross-References

- ▶ Evolutionary Algorithms
- ▶ Evolutionary Computation in Economics
- ▶ Evolutionary Computational Techniques in Marketing
- ▶ Genetic Programming

Recommended Reading

Allen F, Karjalainen R (1999) Using genetic algorithms to find technical trading rules. J Financ Econ 51:245–271

Altman EI (1968) Financial ratios, discriminant analysis and the prediction of corporate bankruptcy. J Financ 23(4):589–609

Andrieu C, de Freitas N, Doucet A, Jordan MI (2003) An introduction to MCMC for machine learning. Mach Learn 50:5–43

Arthur WB (1991) Learning and adaptive economic behavior. Designing economic agents that act like human agents: a behavioral approach to bounded rationality. Am Econ Rev 81:353–359

Arthur WB (1994) Inductive reasoning and bounded rationality: the El Farol problem. Am Econ Rev 84:406–411

Arthur WB, Holland JH, LeBaron B, Palmer RG, Talyer P (1997) Asset pricing under endogenous expectations in an artificial stock market. In: Arthur WB, Durlauf S, Lane D (eds) The economy as an evolving complex system II. Addison-Wesley, Reading

Atiya AF (2001) Bankruptcy prediction for credit risk using neural networks: a survey and new results. IEEE Trans Neural Netw 12(4):929–935

Bak P, Paczuski M, Shubik M (1997) Price variations in a stock market with many agents. Physica A 246:430–453

Barro RJ (2009) Rare disasters, asset prices, and welfare costs. Am Econ Rev 99(1):243–264

Baum LE, Petrie T, Soules G, Weiss N (1970) A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. Ann Math Stat 41:164–171

Bhattacharyya S, Pictet OV, Zumbach G (2002) Knowledge-intensive genetic discovery in foreign exchange markets. IEEE Trans Evol Comput 6(2):169–181

Black F, Scholes M (1973) The pricing of options and corporate liabilities. J Political Econ 81: 637–654

Brabazon A, O'Neill M (2004) Evolving technical trading rules for spot foreign-exchange markets using grammatical evolution. Comput Manag Sci 1(3):311–327

Brabazon A, O'Neill M (eds) (2008) Natural computing in computational finance. Volume 100 of studies in computational intelligence. Springer, Berlin

Brabazon A, O'Neill M (eds) (2009) Natural computing in computational finance, vol 2. Volume 185 of studies in computational intelligence. Springer, Berlin

Brandt MW (2009) Portfolio choice problems. Handb Financ Econom 1:269–336

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. Wadsworth International Group, Belmont

Challet D, Marsili M, Zhang Y-C (2000) Modeling market mechanism with minority game. Physica A 276:284–315

Challet D, Zhang Y-C (1997) Emergence of cooperation and organization in an evolutionary game. Physica A 246:407

Chen S-H (ed) (2002a) Evolutionary computation in economics and finance. Volume 100 of studies in fuzziness and soft computing. Springer, New York/Secaucus

Chen S-H (ed) (2002b) Genetic algorithms and genetic programming in computational finance. Kluwer Academic Publishers, Norwell

Chen S-H, Lu C-F (1999) Would evolutionary computation help in designs of artificial neural nets in forecasting financial time series? In: Proceeding of 1999 congress on evolutionary computation, Washington, DC. IEEE Press, pp 275–280

Chen S-H, Wang H-S, Zhang B-T (1999) Forecasting high-frequency financial time series with evolutionary neural trees: the case of hang-seng stock index. In: Arabnia HR (ed) Proceedings of the international conference on artificial intelligence, IC-AI'99, Las Vegas, vol 2, 28 June–1 July 1999. CSREA Press, pp 437–443

Chen S-H, Wang PP (eds) (2004) Computational intelligence in economics and finance. Advanced information processing. Springer, Berlin/New York

Chen S-H, Wang PP, Kuo T-W (eds) (2007) Computational intelligence in economics and finance, volume II. Advanced information processing. Springer, Berlin/Heidelberg

Chen S-H, Yeh C-H (2001) Evolving traders and the business school with genetic programming: a new architecture of the agent-based artificial stock market. J Econ Dyn Control 25(3–4):363–393

Chen S-H, Yeh C-H, Lee W-C (1998) Option pricing with genetic programming. In: Koza JR, Banzhaf W, Chellapilla K, Deb K, Dorigo M, Fogel DB, Garzon MH, Goldberg DE, Iba H, Riolo R (eds) Genetic programming 1998: proceedings of the third annual conference, University of Wisconsin, Madison, 22–25 July 1998. Morgan Kaufmann, pp 32–37

Chidambaran NK, Triqueros J, Jevons Lee C-W (2002) Option pricing via genetic programming. In: Chen S-H (ed) Evolutionary computation in economics and finance. Volume 100 of studies in fuzziness and soft computing, chapter 20. Physica Verlag, New York, pp 383–398

Cincotti S, Ponta L, Raberto M (2005) A multi-assets artificial stock market with zero-intelligence traders. In: WEHIA 2005 (13–15 June 2005), Essex

Coello CA (2006) Evolutionary multi-objective optimization and its use in finance. MIMEO, CINVESTAV-IPN, Mexico

Dempster MAH, Payne TW, Romahi Y, Thompson GWP (2001) Computational learning techniques for intraday FX trading using popular technical indicators. IEEE Trans Neural Netw 12:744–754

Diosan L (2005) A multi-objective evolutionary approach to the portfolio optimization problem. In: CIMCA'05: proceedings of the international conference on computational intelligence for modelling, control and automation and international conference on intelligent agents, web technologies and internet commerce vol-2 (CIMCA-IAWTIC'06), Washington, DC. IEEE Computer Society, pp 183–187

Doerner K, Gutjahr WJ, Hart RF, Strauss C, Stummer C (2004) Pareto ant colony optimization: a metaheuristic approach to multiobjective portfolio selection. Ann Oper Res 131:79–99

Doucet A, de Freitas N, Gordon NJ (2001) An introduction to sequential Monte Carlo methods. In: Doucet A, de Freitas N, Gordon NJ (eds) Sequential Monte Carlo methods in practice. Springer, New York, pp 1–13

Dutta S, Shekhar S (1988) Bond rating: a nonconservative application of neural networks. IEEE Int Conf Neural Netw 2:443–450

Fan K, Brabazon A, O'Sullivan C, O'Neill M (2007) Option pricing model calibration using a real-valued quantum-inspired evolutionary algorithm. In: GECCO'07: proceedings of the 9th annual conference on genetic and evolutionary computation. ACM, New York, pp 1983–1990

Fernández F, Isasi P (2004) Evolutionary design of nearest prototype classifiers. J Heuristics 10(4): 431–454

Fisher RA (1936) The use of multiple measurements in taxonomic problems. Ann Eugen 7:179

Gabaix X (2012) Variable rare disasters: an exactly solved framework for ten puzzles in macro-finance. Q J Econ 127(2):645–700

Garcia-Almanza AL, Tsang EPK (2008) Evolving decision rules to predict investment opportunities. Int J Autom Comput 5(1):22–31

Ghandar A, Michalewicz Z, Schmidt M, To TD, Zurbrugg R (2008) Computational intelligence for evolving trading rules. IEEE Trans Evol Comput 13(1):71–86

Gode DK, Sunder S (1992) Allocative efficiency of markets with zero intelligence (z1) traders: market as a partial substitute for individual rationality. GSIA working papers 1992-16, Tepper School of Business, Carnegie Mellon University

Gordon NJ, Salmond DJ, Smith AFM (1993) Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In: IEE Proceedings F (Radar and Signal Processing), vol 140, IET, pp 107–113

Gourio F (2008) Disasters and recoveries. Am Econ Rev 98:68–73

Grothmann R (2002) Multi-agent market modeling based on neural networks. PhD thesis, Faculty of Economics, University of Bremen

Hassan G, Clack CD (2008) Multiobjective robustness for portfolio optimization in volatile environments. In: GECCO'08: proceedings of the 10th annual conference on Genetic and evolutionary computation. ACM, New York, pp 1507–1514

Hodrick RJ, Prescott EC (1997) Postwar us business cycles: an empirical investigation. J Money Credit Bank 29:1–16

Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor

Holland JH, Miller JH (1991) Artificial adaptive agents in economic theory. Am Econ Rev 81:365–370

Huang C-L, Chen M-C, Wang C-J (2007) Credit scoring with a data mining approach based on support vector machines. Expert Syst Appl 33(4): 847–856

Huang Z, Chen H, Hsu C-J, Chen W-H, Wu S (2004) Credit rating analysis with support vector machines and neural networks: a market comparative study. Decis Support Syst 37(4):543–558

Hull J (2008) Options, futures and other derivatives. Prentice Hall series in finance. Prentice Hall, Upper Saddle River

Huo Y, Cai Z, Gong W, Liu Q (2014) A new adaptive Kalman filter by combining evolutionary algorithm and fuzzy inference system. In: 2014 IEEE congress on evolutionary computation (CEC), Beijing, pp 2893–2900

Jazwinski AH (1970) Stochastic processes and filtering theory. Academic Press, New York

Julier SJ, Uhlmann JK (1997) A new extension of the Kalman filter to nonlinear systems. In: International symposium on aerospace/defense sensing, simulation and controls, Orlando, vol 3, pp 182–193

Kalman RE (1960) A new approach to linear filtering and prediction problems. J Fluids Eng 82(1):35–45

Kirman AP (1992) Whom or what does the representative individual represents? J Econ Perspect 6: 117–136

Kumar KS, Dustakar NR, Jatoth RK (2010) Evolutionary computational tools aided extended Kalman filter for ballistic target tracking. In: 2010 3rd international conference on emerging trends in engineering and technology (ICETET), Goa, pp 588–593

Kumar S, Thulasiram RK, Thulasiraman P (2008) A bioinspired algorithm to price options. In: C3S2E'08: proceedings of the 2008 C3S2E conference. ACM, New York, pp 11–22

LeBaron B (2001) A builder's guide to agent based financial markets. Quant Financ 1:254–261

LeBaron B (2006) Agent-based computational finance. In: Judd KL, Tesfatsion L (eds) Handbook of computational economics, volume 2: agent-based computational economics. Handbooks in economics, chapter 24. North-Holland, pp 1187–1234

Levy M, Levy H, Solomon S (1994) A microscopic model of the stock market: cycles, booms and crashes. Econ Lett 45:103–111

Lipinski P (2004) Evolutionary data-mining methods in discovering stock market expertise from financial time series. PhD thesis, University of Wroclaw, Wroclaw

Lux T (1998) The socio-economic dynamics of speculative markets: interacting agents, chaos, and the fat tails of return distributions. J Econ Behav Organ 33:143–165

Lux T, Ausloos M (2002) Market fluctuations I: scaling, multiscaling and their possible origins. In: Bunde A, Kropp J, Schellnhuber HJ (eds) Theories of disaster – scaling laws governing weather, body, and stock market dynamics. Springer, Berlin Heidelberg pp 373–409

Maringer D (2005) Portfolio management with heuristic optimization. Volume 8 of advances in computational management science. Springer Dordrecht, The Netherlands

Maringer D (2006) Small is beautiful: diversification with a limited number of assets. Working paper WP005-06, Centre for Computational Finance and Economic Agents, University of Essex

Martens D, Baesens B, Gestel TV, Vanthienen J (2007) Comprehensible credit scoring models using rule extraction from support vector machines. Eur J Oper Res 183(3):1466–1476

Martinez-Jaramillo S, Tsang EPK (2009a) Evolutionary computation and artificial financial markets. In: Natural computing in computational finance. Volume 185 of studies in computational intelligence. Springer, Berlin/Heidelberg, pp 137–179

Martinez-Jaramillo S, Tsang EPK (2009b) An heterogeneous, endogenous and coevolutionary gp-based financial market. IEEE Trans Evol Comput 13:33–55

Ong C-S, Huang J-J, Tzeng G-H (2005) Building credit scoring models using genetic programming. Expert Syst Appl 29(1):41–47

O'Sullivan C (2007) Parameter uncertainty in Kalman filter estimation of the cir term structure model. Centre for Financial Markets working paper series

WP-07-18, Centre for Financial Markets, School of Business, University College Dublin

Paleologo G, Elisseeff A, Antonini G (2010) Subagging for credit scoring models. Eur J Oper Res. 201(2):490–499

Potvin J-Y, Soriano P, Vallée M (2004) Generating trading rules on the stock markets with genetic programming. Comput Oper Res 31(7):1033–1047

Quintana D, Saez Y, Mochon A, Isasi P (2008) Early bankruptcy prediction using enpc. Appl Intell 29(2):157–161

Rezaei N, Kordabadi H, Elkamel A, Jahanmiri A (2008) An optimal extended Kalman filter designed by genetic algorithms. Chem Eng Commun 196(5):602–615

Ross SA (1976) The arbitrage theory of capital asset pricing. J Econ Theory 13(3):341–360

Schoreels C, Logan B, Garibaldi JM (2004) Agent based genetic algorithm employing financial technical analysis for making trading decisions using historical equity market data. In: IAT'04: proceedings of the intelligent agent technology, IEEE/WIC/ACM international conference, Washington, DC. IEEE Computer Society, pp 421–424

Sharpe WF Capital asset prices: a theory of market equilibrium under conditions of risk*. J Financ 19(3):425–442 (1964)

Shin K-S, Lee Y-J (2002) A genetic algorithm application in bankruptcy prediction modeling. Expert Syst Appl 23(3):321–328

Simon HA (1982) Models of bounded rationality, vol 2. MIT Press, Cambridge, MA

Streichert F, Ulmer H, Zell A (2004) Evaluating a hybrid encoding and three crossover operators on the constrained portfolio selection problem. In: Proceedings of the 2004 congress on evolutionary computation. IEEE Press, pp 932–939

Sung TK, Chang N, Lee G (1999) Dynamics of modeling in data mining: interpretive approach to bankruptcy prediction. J Manag Inf Syst 16(1): 63–85

Tesfatsion L (2002) Agent-based computational economics: growing economies from the bottom up. Artif Life 8:55–82

Tsang EPK, Martinez-Jaramillo S (2004) Computational finance. In: IEEE computational intelligence society newsletter. 3(8):8–13

Tsang EPK, Yung P, Li J (2004) Eddie-automation, a decision support tool for financial forecasting. J Decis Support Syst Spec Issue Data Min Financ Decis Mak 37(4):559–565

Turku BB, Back B, Laitinen T, Sere K, Wezel MV (1996) Choosing bankruptcy predictors using discriminant analysis, logit analysis, and genetic algorithms. In: Proceedings of the first international meeting on artificial intelligence in accounting, finance and tax, p 337356

West D (2000) Neural network credit scoring models. Comput Oper Res 27(11–12):1131–1152

Yang J (2002) The efficiency of an artificial double auction stock market with neural learning agents. In Evol Comput Econ Financ 85–106, Physica-Verlag Heidelbergh New York

Yin Z, Brabazon A, O'Sullivan C (2007) Adaptive genetic programming for option pricing. In: GECCO'07: proceedings of the 2007 GECCO conference companion on genetic and evolutionary computation. ACM, New York, pp 2588–2594

Zhou Q, Lin C, Yang W (2006) Multi-classifier combination for banks credit risk assessment. In: 1st IEEE conference on industrial electronics and applications, pp 1–4

Evolutionary Computational Techniques in Marketing

Alma Lilia García-Almanza[1], Biliana Alexandrova-Kabadjova[2], and Serafín Martínez-Jaramillo[3]
[1]Directorate of Regulation and Supervision, Banco de México, Mexico City, Mexico
[2]Banco de México, Mexico City, Mexico
[3]Directorate of Financial System Risk Analysis, Banco de México, Mexico City, Mexico

Motivation and Background

The Internet and social networks are key factors that have strongly affected market competition, as they provide customers with more choice in products, services, and prices. For instance, well-established electronic commerce companies such as Amazon, Booking, TripAdvisor, and others provide rankings of their products based on past customer reviews. In the same vein, social networks are a powerful tool to spread good or bad comments about products or services, and they can directly influence potential clients, since the members of the same social network usually share interests and have similar economic levels. For those reasons, marketing teams have focused efforts on creating intelligent business strategies. New artificial intelligence approaches to marketing have emerged, especially evolutionary algorithms used to solve a variety of marketing problems such as the design of attractive products and services for consumers, the analysis of populations or social networks to target potential clients, the design of new marketing strategies, and more. Nowadays, a huge amount of data on almost any kind of human activity has been stored in structured and unstructured forms. The data is a gold mine, the analysis of which can provide useful information for competing more efficiently in the market. For that reason, machine learning techniques have been used to discover useful patterns for creating user-friendly interfaces and new market segments, among other aims. Many evolutionary computational techniques have been applied to marketing problems in order to obtain a commercial advantage over competitors.

Applications

Marketing is a very dynamic area, as it evolves alongside technology and aims to keep promoted products alive in the market.

The Design of New Products

One of the goals of marketing is to discover products of superior value and quality. To achieve this goal in Fruchter et al. (2006), the authors propose to design a product line rather than a single product. The authors argue that by offering a product line, the manufacturer can customize products according to the needs of different market niches, which would result in higher customer satisfaction and more buyers. Nevertheless, as the time required by the amount of data on customer preferences increases, the optimization process of the product line becomes very hard to manage. For that reason, the authors applied the use of genetic algorithms (GAs) to solve the problem heuristically, and the performance of each solution was valued according to the manufacturer's profits. In a similar way, Liu and Ong (2008) used a GA to solve a marketing segmentation problem. In this case, the evolutionary algorithm was applied to reach all customers effectively.

In the approach proposed by Sundar Balakrishnan and Jacob (1996), a GA was used to optimize for customer preference in product design. The authors followed a three-step methodology in order to create a new product. First, the set of attributes subject to adjustment, such as color or shape, was determined. Second, customer preferences were collected. Finally, a GA was applied to select those attributes that satisfy a larger number of customers.

Targeting Potential Clients

Bhattacharyya (2000) proposed a GA in combination with a case-based reasoning (CBR) system to predict customer purchasing behavior. The objective was to identify potential customers for a specific product or service. This approach was developed and tested with real cases by direct marketing from a worldwide insurance company. An optimization mechanism was integrated into the classification system in order to select those customers most likely to acquire an insurance.

Advertisement

Advertisement is an important area of marketing. It is defined as the activity of attracting public attention to a product or business. Since personalized advertisement improves marketing efficiency, Kwon and Moon (2001) proposed a personalized prediction model to be used in email marketing. A circuit model combined with genetic programs (GPs) was proposed to analyze customer information. The result was a set of recommended rules. It was tested over a general mass marketing. According to the authors, the model showed a significant improvement in sales. In another approach, Naik et al. (1998) used a GA combined with a Kalman filter procedure to determine the best media schedule for advertisement, which at the time was constrained by a budget. This approach evaluated a large number of alternative media schedules to decide upon an optimal media planning solution. The Internet has become very popular and convenient for offering and purchasing, since many products and services can be found easily in a very short time, increasing competition among providers. Since these kinds of sales do not directly involve human interaction, it is essential to design new and better strategies to personalize Web pages. For instance, Abraham and Ramos (2003) proposed an ant clustering algorithm to discover Web usage patterns and a linear genetic programming to analyze visitor behavior. The objective was to discover useful knowledge from user interactions with the Web. The knowledge was used to design adaptive Web sites, business and support services, personalization, network traffic flow analysis, and more.

According to Scanlon (2008), the company Staples used a software called Affinnova to redesign and relaunch its paper brand. Affinnova was designed by Waltham, and it uses a GA to simulate the evolution of consumer markets where strong products survive and weak ones die out. The strongest possible design emerges after several generations. A panel of 750 consumers selected their favorite options from each generation. The software analyzed customer choices over multiple generations to identify preference patterns. Surveys included consumer profiles that contain basic demographic information, customer beliefs, and consumer habits. Clients can also segment results and understand how different designs appeal to different consumers. Affinnova's research also helped to identify the imagery and messaging that would most appeal to consumers.

To summarize, EC has been used to solve a wide variety of marketing problems. Given that ECs are global optimization methods, they can be applied to forecasting and data mining. In this respect, they have great potential for use in the field of marketing. EC techniques allow for the extraction and analysis of customer patterns among large amounts of data, and forecasts of purchasing tendencies, among many other aims.

Cross-References

- ▶ Evolutionary Algorithms
- ▶ Evolutionary Computation in Economics
- ▶ Evolutionary Computation in Finance
- ▶ Genetic and Evolutionary Algorithms
- ▶ Genetic Programming

Recommended Reading

Abraham A, Ramos V (2003) Web usage mining using artificial ant colony clustering and linear genetic programming. In: Congress on evolutionary computation (CEC), Canberra, vol 2. IEEE, pp 1384–1391

Bhattacharyya S (2000) Evolutionary algorithms in data mining: multi-objective performance modeling for direct marketing. In KDD'00: proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining, New York. ACM, pp 465–473

Fruchter G, Fligler A, Winer R (2006) Optimal product line design: a genetic algorithm approach to mitigate cannibalization. J Optim Theory Appl 131(2):227–244

Kwon Y-K, Moon B-R (2001) Personalized email marketing with a genetic programming circuit model. In: Spector L, Goodman ED, Wu A, Langdon WB, Voigt H-M, Gen M, Sen S, Dorigo M, Pezeshk S, Garzon MH, Burke E (eds) Proceedings of the genetic and evolutionary computation conference (GECCO-2001), San Francisco. Morgan Kaufmann, pp 1352–1358

Liu H-H, Ong C-S (2008) Variable selection in clustering for marketing segmentation using genetic algorithms. Expert Syst Appl 34(1):502–510

Naik PA, Mantrala MK, Sawyer AG (1998) Planning media schedules in the presence of dynamic advertising quality. Mark Sci 17(3):214–235

Scanlon J, (2008) "Staples' Evolution", Bloomberg.com, Bloomberg, <http://www.bloomberg.com/news/articles/2008-12-29/staples-evolutionbusinessweek-business-news-stock-market-and-financial-advice>.

Sundar Balakrishnan PV, Jacob VS (1996) Genetic algorithms for product design. Manag Sci 42(8):1105–1117

Evolutionary Computing

▶ Evolutionary Algorithms

Evolutionary Constructive Induction

▶ Evolutionary Feature Selection and Construction

Evolutionary Feature Selection

▶ Evolutionary Feature Selection and Construction

Evolutionary Feature Selection and Construction

Krzysztof Krawiec
Poznan University of Technology, Poznan,
Poland

Abstract

Representation of input data has an essential influence on the performance of machine learning systems. Evolutionary algorithms can be used to transform data representation by selecting some of the existing features (evolutionary feature selection) or constructing new features from the existing ones (evolutionary feature construction). This entry provides the rationale for both these approaches and systematizes the research and applications in this area.

Synonyms

EFSC; Evolutionary constructive induction; Evolutionary feature selection; Evolutionary feature synthesis; Genetic attribute construction; Genetic feature selection

Definition

Evolutionary feature selection and construction (EFSC) is a bio-inspired methodology for explicit modification of input data of a learning system. It uses evolutionary computation (EC) to construct a mapping from the original data representation space onto a *secondary* representation space. In evolutionary feature selection (EFS), that mapping consists in dropping off some of the features (▸ attributes) from the original representation so that the dimensionality of the resulting representation space is not greater than that of the original space. In evolutionary feature construction (EFC), an evolutionary algorithm creates (synthesizes) new features (derived attributes) that complement and/or replace the original ones. Therefore, EFS may be considered as a special case of EFC.

A typical EFSC algorithm maintains a population of solutions, each of them encoding a specific mapping. The best mapping found in evolutionary search becomes the data preprocessor for the classifier. Usually, EFSC takes place in the training phase only, and the evolved mapping does not undergo further changes in the testing phase.

Though EFSC is technically a form of data preprocessing (see ▸ Data Preparation), some of its variants may as well involve an internal inductive process in the fitness function. Also, EFS and EFC may be considered as special cases of ▸ Feature Selection and ▸ Feature Construction, respectively. EFC is also partially inspired by ▸ Constructive Induction.

Motivation and Background

Real-world machine-learning problems often involve a multitude of attributes, which individually have low informative content and cannot provide satisfactory performance of the learning system. This applies in particular to data-abundant domains like image analysis and signal processing. When faced with many low-quality attributes, induction algorithms tend to build classifiers that perform poorly in terms of classification accuracy. This problem may be alleviated by removing some features from the original representation space (*feature selection*) or introducing new features defined as informative expressions (arithmetic, logical, etc.) built of *multiple* attributes (*feature construction*).

Many learning algorithms lack the ability of discovering intricate dependencies between attributes, which is a necessary precondition for successful feature selection and construction. This gap is filled out by EFSC, which uses EC to get rid of superfluous attributes and to construct new features. Benefits of EFSC are similar to those of general ▸ Feature Selection and ▸ Feature Construction and include reduced dimensionality of the input space, better predictive accuracy of the learning system, faster training and querying, and better readability of the acquired knowledge.

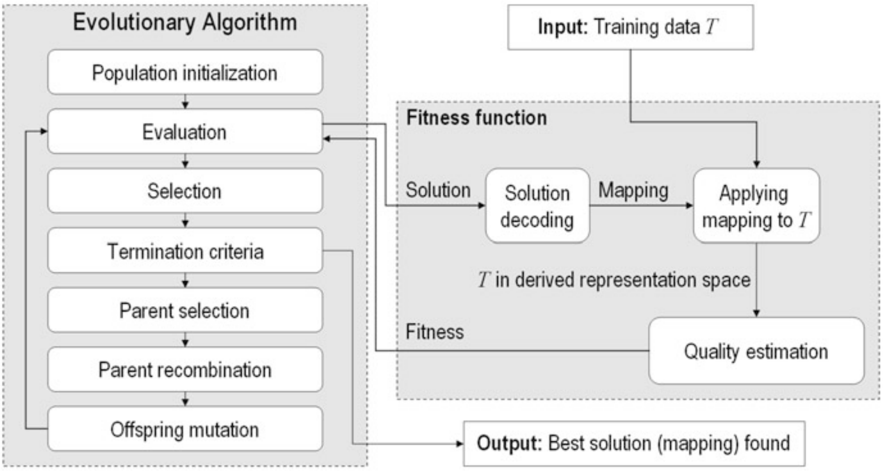
Feature selection and feature construction may be conveniently formulated as an optimization problem with each solution corresponding to a particular feature subset (for feature selection) or to a particular definition of new features (for feature construction). The number of such solutions grows exponentially with the number of original features, rendering the exact search methods infeasible. EC techniques are particularly well-suited to heuristically search these solution spaces. They do not make any assumptions about the optimized function (in contrast to, e.g., the branch-and-bound algorithm) and perform global heuristic search, typically finding high-quality solutions in acceptable time. These virtues are important in EFSC, where the objective function depends on the training data, and it is difficult to predict its properties.

Another strength of EC is easy tailoring to a given task. For instance, a subset of features in EFS is usually encoded as a bit-string solution in genetic algorithm (GA), where a bit at a particular position determines the selection or exclusion of the corresponding feature (Vafaie and Imam 1994; Yang and Honavar 1998). In EFC, definitions of constructed features may be conveniently represented as genetic programming (GP) expressions (Rizki et al. 2002; Teller and Veloso 1997). Also, an evolutionary algorithm naturally produces *many* solutions. This makes it a convenient tool for, e.g., parallel construction of multiple representations (feature subsets) that may be subsequently used in a compound classifier.

Structure of Learning System

Typically, EFSC uses a variant of evolutionary algorithm (usually GA for EFS or genetic programming for EFC) to maintain a population of solutions (individuals), each of them encoding a particular subset of features (for EFS) or definition of new features (for EFC). Solutions undergo mutations, recombinations, and selection. Selective pressure is exerted by a fitness function that estimates a solution's quality by analyzing selected properties of the secondary representation space (see Fig. 1). This usually involves three steps:

1. *Decoding* of solution (retrieving the mapping from the encoded solution).
2. *Transforming* the training set into the secondary representation space according to the mapping.
3. Estimating the *quality* of the secondary representation space, which becomes a solution's fitness.



Evolutionary Feature Selection and Construction, Fig. 1 Evolutionary feature selection and construction

The quality measures employed in step 3 may be grouped into two categories. *Filter approach* relies on the measures that characterize the desired properties of training data in the secondary space (e.g., class separability), abstracting from any particular induction algorithm. *Wrapper approach* estimates the predictive ability in the secondary representation space by a *specific* induction algorithm, usually by partitioning the training set into several subsets and performing multiple train-and-test experiments (e.g., cross-validation). The wrapper approach, though computationally more expensive, takes into account the inductive and representational biases of the employed induction algorithm and thanks to that often proves superior in terms of classification accuracy.

The result of a typical EFSC procedure is the best solution found in an evolutionary run, i.e., the most fit representation mapping. This mapping serves as a preprocessor of input data and is subsequently used to induce the final classifier from the training set. The trained classifier together with the preprocessing provided by the mapping is the final outcome of the EFSC-enriched training process and may be used for classification of new examples.

EFS is the simplest variant of EFSC. In this case, a solution encodes the indices of attributes that should remain in the resulting secondary representation. This leads to straightforward encoding characteristic for GA, with each solution being a bit string as long as the number of original attributes. EFS may be thus easily implemented using off-shelf EA software packages. More sophisticated EFS approaches have been also considered, like evolving GP individuals that *rank* or *score* features (Zhang and Rockett 2011).

Evolutionary feature weighting (EFW) is a direct generalization of EFS, where the evolutionary search weighs the features instead of selecting them. Solutions in EFW are real-valued vectors. EFW requires a wrapper fitness function that can take attribute weights into account. In Komosiński and Krawiec (2000), EFW has been used with a nearest neighbor-based wrapper fitness function to weigh features for a medical diagnosing problem.

EFC usually employs genetic programming to represent feature transformation. Each GP solution encodes an expression tree that uses the original attributes and numeric constants as leaves (terminals) and functions from a predefined vocabulary as internal tree nodes (nonterminals). The value returned by such an expression when applied to an example is interpreted as the new feature. Function set usually encompasses simple arithmetics and elementary functions. The evolved features replace or extend the original ones. If a single new feature is insufficient to provide satisfactory discriminative ability, several GP trees can be encoded within each solution.

EFC is particularly useful in image analysis and computer vision tasks, which naturally tend to involve large numbers of attributes. In such contexts, an EFC algorithm evolves GP solutions that construct higher-level features from low-level image attributes (Krawiec and Bhanu 2005) or implement advanced feature detectors (Howard et al. 2006; Puente et al. 2009). Alternatively, solutions encode chains of operations that process the entire image globally according to the goal specified by the fitness function. Other representations of EFC solutions have been studied as well in GP, including, e.g., graphs (Teller and Veloso 1997) or sequences of operations (Bhanu et al. 2005).

It has been demonstrated that an EFC task may be decomposed into several semi-independent subtasks using cooperative coevolution, a variant of evolutionary algorithm that maintains several populations hosting individuals that encode partial solutions (Krawiec and Bhanu 2005). Other work demonstrates that fragments of GP expressions encoding feature definitions may help to discover good features in other learning tasks (Jaśkowski, Krawiec, and Wieloch 2007).

Applications

Real-world applications of EFSC are numerous and include medical and technical diagnosing, genetics, detection of intrusions in computer networks, air quality forecasting, brain-computer interfaces, seismography, robotics, face recognition, handwriting recognition, vehicle detection in visual, infrared, and radar modality, image segmentation, satellite imaging, and stereovision. EFS has been built into several machine learning and neural network software packages (e.g., WEKA, Statistica). A ready-to-use implementation of EFC is available in RapidMiner; alternatively, it can be facilitated with the existing EC frameworks like ECJ (http://cs.gmu.edu/~eclab/projects/ecj/). More examples of real-world applications of EFSC may be found in Langdon et al. (2009).

Future Directions

Nowadays, EFC becomes more and more unified with GP-based classification and regression, where solutions are expected to perform the complete classification or regression task rather than to implement only feature definitions. Recently, EFSC has also witnessed the growing popularity of the *multiobjective* evolutionary techniques. In EFC, it is now common to include the complexity of feature definition (reflected by program size in GP) as an additional objective alongside the accuracy of classification (Neshatian and Zhang 2011). This is intended to reduce the so-called *program bloat* (the excessive growth of programs that often pesters GP systems) and so curtail overfitting, because complex features are less likely to generalize well. Other studies involve more "helper objectives," like Bayes error estimate (Olague and Trujillo 2012) or Fisher criterion. Domain-specific measures are also occasionally employed in this character. For instance, in a computer vision study (Arnaldo et al. 2014), interest point detectors are evolved using three objectives that capture detector's stability, spatial dispersion of detected points, and their information content.

The online genetic programming bibliography (Langdon et al. 2009) covers most of the works in evolutionary feature selection and construction. A concise review of contemporary GP research involving feature construction for image analysis and object detection may be found in Krawiec et al. (2007). A systematization of different evolutionary approaches to feature construction is also presented in Bhanu et al. (2005).

Cross-References

- ▶ Constructive Induction
- ▶ Data Preparation
- ▶ Feature Selection

Recommended Reading

Arnaldo I, Krawiec K, O'Reilly U-M (2014) Multiple regression genetic programming. In: Igel C, Arnold DV, Gagne C, Popovici E, Auger A, Bacardit J, Brockhoff D, Cagnoni S, Deb K, Doerr B, Foster J, Glasmachers T, Hart E, Heywood MI,Iba H, Jacob C, Jansen T, Jin Y, Kessentini M, Knowles JD, Langdon WB, Larranaga P, Luke S, Luque G, McCall JAW, Montes de Oca MA, Motsinger-Reif A, Ong YS, Palmer M, Parsopoulos KE, Raidl G, Risi S, Ruhe G, Schaul T, Schmickl T, Sendhoff B, Stanley KO, Stuetzle T, Thierens D, Togelius J, Witt C, Zarges C (eds) GECCO '14: proceedings of the 2014 conference on genetic and evolutionary computation, SIGEVO, Vancouver, 12–16 July. ACM, New York, pp 879–886. doi:10.1145/2576768.2598291, ISBN 978-1-4503-2662-9, http://doi.acm.org/10.1145/2576768.2598291

Bhanu B, Lin Y, Krawiec K (2005) Evolutionary synthesis of pattern recognition systems. Springer, New York

Howard D, Roberts SC, Ryan C (2006) Pragmatic genetic programming strategy for the problem of vehicle detection in airborne reconnaissance. Pattern Recognit Lett 27(11):1275–1288

Jaśkowski W, Krawiec K, Wieloch B (2007) Knowledge reuse in genetic programming applied to visual learning. In: Thierens D et al (eds) GECCO'07: proceedings of the 9th annual conference on genetic and evolutionary computation, vol 2. ACM Press, London, pp 1790–1797

Komosiński M, Krawiec K (2000) Evolutionary weighting of image features for diagnosing of CNS tumors. Artif Intell Med 19(1):25–38

Krawiec K, Bhanu B (2005) Visual learning by coevolutionary feature synthesis. IEEE Trans Syst Man Cybern Part B 35(3):409–425

Krawiec K, Howard D, Zhang M (2007) Overview of object detection and image analysis by means of genetic programming techniques. In: Proceedings of frontiers in the convergence of bioscience and information technologies 2007 (fbit2007), Jeju, 11–13 oct 2007. IEEE CS Press, pp 779–784

Langdon W, Gustafson S, Koza J (2009) The genetic programming bibliography. http://www.cs.bham.ac.uk/~wbl/biblio/ [online]

Neshatian K, Zhang M (2011) Using genetic programming for context-sensitive feature scoring in classification problems. Connect Sci 23(3):183–207. doi:10.1080/09540091.2011.630065, http://www.tandfonline.com/doi/abs/10.1080/09540091.2011.630065, http://www.tandfonline.com/doi/pdf/10.1080/09540091.2011.630065

Olague G, Trujillo L (2012) Interest point detection through multiobjective genetic programming. Appl Soft Comput 12(8):2566–2582. doi:10.1016/j.asoc.2012.03.058, ISSN 1568-4946, http://www.sciencedirect.com/science/article/pii/S1568494612001706

Puente C, Olague G, Smith SV, Bullock SH, González-Botello MA, Hinojosa-Corona A (2009) A novel GP approach to synthesize vegetation indices for soil eros ion assessment. In: Giacobini M et al (eds) Applications of evolutionary computing. Springer, Berlin/New York, pp 375–384

Rizki MM, Zmuda MA, Tamburino LA (2002) Evolving pattern recognition systems. IEEE Trans Evolut Comput 6(6):594–609

Teller A, Veloso M (1997) PADO: a new learning architecture for object recognition. In: Ikeuchi K, Veloso M (eds) Symbolic visual learning. Oxford Press, New York, pp 77–112

Vafaie H, Imam IF (1994) Feature selection methods: genetic algorithms vs. greedy-like search. In: Proceedings of international conference on fuzzy and intelligent control systems, Louisville, Mar 1994

Yang J, Honavar V (1998) Feature subset selection using a genetic algorithm. IEEE Trans Intell Syst 13(2):44–49

Zhang Y, Rockett PI (2011) A generic optimising feature extraction method using multiobjective genetic programming. Appl Soft Comput 11(1):1087–1097. doi:10.1016/j.asoc.2010.02.008, ISSN 1568-4946, http://www.sciencedirect.com/science/article/B6W86-4YGHGKT-2/2/3c6f14d2e029af14747957a5a2ccfd11

Evolutionary Feature Synthesis

▶ Evolutionary Feature Selection and Construction

Evolutionary Fuzzy Systems

Carlos Kavka
University of Trieste, Trieste, Italy

Definition

An evolutionary fuzzy system is a hybrid automatic learning approximation that integrates ▶ fuzzy systems with ▶ evolutionary algorithms, with the objective of combining the optimization and learning abilities of evolutionary algorithms together with the capabilities of fuzzy systems to deal with approximate knowledge. Evolutionary fuzzy systems allow the optimization of the knowledge provided by the expert in terms of linguistic variables and fuzzy rules, the generation of some of the components of fuzzy systems based on the partial information provided by the expert, and in some cases even the generation of fuzzy systems without expert information. Since many evolutionary fuzzy systems are based on the use of genetic algorithms, they are also known as *genetic fuzzy systems*. However, many models presented in the scientific literature also use genetic programming, evolutionary programming, or evolution strategies, making the term *evolutionary fuzzy systems* more adequate. Highly related is the concept of *evolutionary neuro-fuzzy systems*, where the main difference is that the representation is based on neural networks. Recently, the related concept of *evolving fuzzy systems* has been introduced, where the main objective is to apply evolutionary techniques to the design of fuzzy systems that are adequate to the control of nonstationary processes, mainly on real-time applications.

Motivation and Background

One of the most interesting properties of a fuzzy system is its ability to represent expert knowledge by using linguistic terms of everyday common use, allowing the description of uncertainty, vagueness, and imprecision in the expert knowledge. The linguistic terms, which are imprecise by their own nature, are, however, defined very precisely by using fuzzy theory concepts.

The usual approach to build a fuzzy system consists in the definition of the membership functions and the rule base in terms of expert knowledge. Compared with other rule-based approaches, the process of extracting knowledge from experts and representing it formally is simpler, since linguistic terms can be defined to match the terms used by the experts. In this way, rules are defined establishing relations between the input and output variables using these linguistic terms. However, even if there is a clear advantage of using the terms defined as ▸ fuzzy sets, the knowledge extraction process is still difficult and time consuming, usually requiring a very difficult manual fine tuning process. It should be noted that no automatic framework to determine the parameters of the components of the fuzzy system exists yet, generating the need for methods that provide adaptability and learning ability for the design of fuzzy systems.

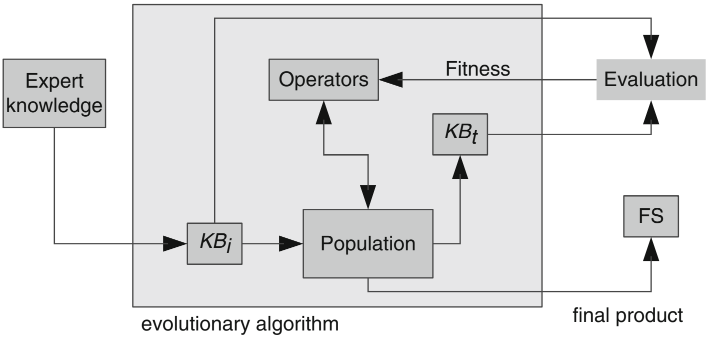
Since it is very easy to map a fuzzy system into a feedforward neural network structure, it is not surprising that many methods based on neural network learning have been proposed to automate the fuzzy system building process (Hoffmann 2001; Karr and Gentry 1993). The combined approach provides advantages from both worlds: the low level learning and computational power of neural networks is joined together with the high level human-like thinking and reasoning of fuzzy systems. However, this approach can still face some problems, such as the potential risk of its learning algorithms to get trapped in local minimum, the possible need for restriction of the membership functions to follow some mathematical properties (like differentiability), and the difficulties of inserting or extracting knowledge in some approaches, where the obtained linguistic terms can exhibit a poor semantic due to the usual black-box processing of many neural networks models.

Evolutionary algorithms provide a set of properties that make them ideal candidates for the optimization and design of fuzzy systems, and in fact, there are many methods that have been proposed in the literature to design or tune the different components of fuzzy systems. Evolutionary systems exhibit robust performance and global search characteristics, while requiring only a simple quality measure from the environment. There is no need for gradient information or input/output patterns. Other strengths come from its parallel nature: instead of selecting a single solution and refining it, in most evolutionary methods, a set of alternative solutions is considered and evolved in parallel.

Structure of the Learning System

The learning process defined by an evolutionary fuzzy system starts from the knowledge provided by the expert, which can include all or just some of the components of the knowledge base of a fuzzy system. The evolutionary algorithm that is behind this learning approach can perform the optimization of all the parameters that are provided by the expert, plus the generation of the missing components of the fuzzy system based on the partial specifications provided by the expert.

The model shown in Fig. 1 presents a general architecture of the learning and optimization process in evolutionary fuzzy systems. An initial knowledge base KB_i is built based on the knowledge provided by the expert. Note that KB_i could be (and usually is) a incompletely specified knowledge base. Based on this initial expert knowledge, the evolutionary algorithm creates a population of individuals, which can represent complete fuzzy systems or just a few components of them. The evaluation of the individuals is performed by creating a temporary knowledge base KB_t, which can also be complete or not. By using the information in KB_t, combined with the initial knowledge base KB_i, the individuals are evaluated by determining the error in the approximation of patterns if there are examples available, computing the reinforcement signal (typical situation in control problems), or in any other way depending on the problem characteristics (Babuska 1998; Cordon et al. 2004). The result



Evolutionary Fuzzy Systems, Fig. 1 The general model of the evolutionary fuzzy systems learning and optimization

of the evaluation is typically a single fitness measure, which provides the necessary information for the selection and the variational operators of the evolutionary algorithm. These operators, which can be standard or defined specifically for the problem, combine and mute the individuals based on the fitness value and their specific parameters. The process is repeated till a predefined criterion is fulfilled, obtaining as a final result the fuzzy system *FS*.

Depending on the information provided by the expert, the learning or optimization process performed by the evolutionary fuzzy system can be applied to the database, the fuzzy rule base or both of them. These three approaches are described below.

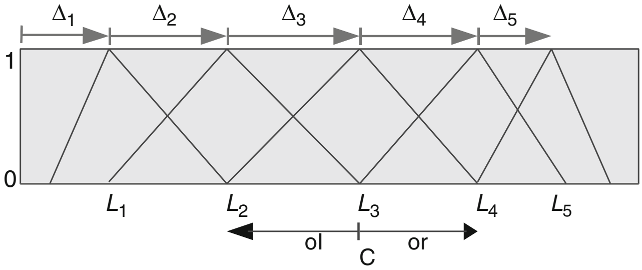
Optimization and Learning of the Fuzzy Database

In this case, it is assumed that the fuzzy rule base is known and provided by the expert. The initial knowledge base KB_i contains the fuzzy rule base, and if provided, the initial approximation of the parameters of antecedents and/or consequents. Since the expert has to define the rule base, and in order to do that, he/she needs to know the labels of the linguistic terms used for the antecedents and consequents, it is usual that the number of fuzzy sets is predefined and kept constant during the evolution.

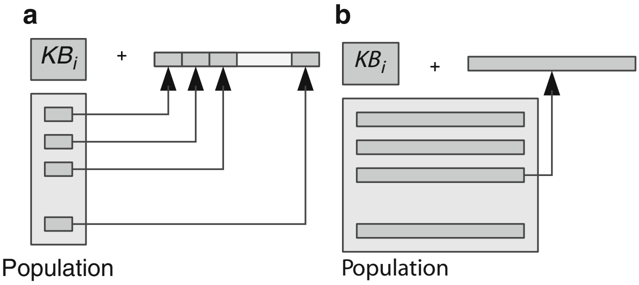
The representation of the individuals contains only the parameters of the fuzzy sets associated to the input linguistic variables, and the fuzzy sets associated to the output variables in the case of a Mamdani fuzzy system, or the associated lineal approximators in the case of a Takagi-Sugeno fuzzy system. Other parameters could also be specified if necessary (scale factors, etc.). Usually, individuals are represented as a fixed length string that is defined as the concatenation of all parameters of the input and output fuzzy sets or approximators. Of course, the representation for the fuzzy sets depends on their particular class: for example, three values are required to represent triangular fuzzy sets, four values to represent trapezoidal fuzzy sets, and two for sigmoidal fuzzy sets. As an example, Fig. 2 shows that three values are necessary to represent a triangular fuzzy set: the center, the left width, and the right width, labeled as c, ol, and od, respectively. From this example, it can be seen that 15 values are required in order to represent the 5 fuzzy sets associated to this single linguistic variable.

However, it is usual to apply fuzzy logic concepts (Zadeh 1988) to simplify the representation, with the implied reduction in the search space, and also, to en- hance the interpretability (Casillas et al. 2003) of the resulting fuzzy system. As an example, it is desirable that the partition associated to a linguistic variable fulfills the completeness property, which establishes that for each point in the input domain, the summation of the membership values of all membership functions must be equal to 1. It is also desirable that the position of the fuzzy sets remains always the same during the evolution, for example in Fig. 2, it means that it is expected that the fuzzy set L_1 will be always at the left of L_2, L_2 always at the left of L_3, and so on. A representation that considers these two requirements can be defined by representing the whole partition specifying the distance from the center of a fuzzy set to the

Evolutionary Fuzzy Systems, Fig. 2 A linguistic variable represented with five fuzzy sets



Evolutionary Fuzzy Systems, Fig. 3 The evaluation of individuals in the (**a**) Michigan and (**b**) Pittsburgh approaches



center of the next one (Hoffmann 2001). The representation of five fuzzy sets then requires only five values (labeled in the figure as Δ_i), which reduces largely the search space and keeps the order of fuzzy sets, while fulfilling the completeness property. Most implementations use real values to represent the parameters.

The operators of the evolutionary algorithm can be standard operators or can be defined specifically based on the selected representation. As an example, operators that modify the width of fuzzy sets, shift the centers, or perform other operations on the fuzzy set representations, linear approximators, or other parameters have been defined in the scientific literature.

Optimization and Learning of the Fuzzy Rule Base

In this case, the fuzzy rule base is not known, or only an initial approximation to it is provided. The other parameters of the knowledge base are known and provided by the expert. The three most usual approximations are

1. Michigan approximation: Each individual of the population codifies a single rule (Bonarini 1996), which means that each individual by itself cannot represent a complete solution to the problem. The knowledge base for evaluation KB_t is built based on the information defined in KB_i and the rules defined by all the individuals from the population combined together (see Fig. 3a). Rules are penalized or rewarded based on its performance during the evaluation. The fuzzy system is then built through the competition of a set of independent rules that have to be learned to collaborate during the evolution.
2. Pittsburgh approximation: Each individual represents the complete rule base. If dynamic creation and removal of rules is allowed, it is necessary to define special variational operators to deal with variable length individuals. Compared with the Michigan approach the evaluation is simpler, since by just combining each individual with KB_i it is possible to build KB_t for evaluation (see Fig. 3b). However, usually, the search space is larger when compared with the Michigan approach.
3. Iterative approximation: Each individual codifies a single rule (Cordon et al. 2001) like in the Michigan approach. However, in each iteration of the algorithm, only the best rule is selected discarding all the others. This selection is based by considering the properties of the rule, such as for example, its covering

degree on a set of examples. The algorithm is then competitive and not cooperative. It is usually necessary to apply algorithms to refine the fuzzy rule set obtained at the end of the evolutionary process, which can include operations, such as for example, the removal of similar rules.

The representation in all of these approximations usually consists of individuals that contain references to the fuzzy sets already defined in KB_i. The representation of each individual can be a sequence of integers where each one is an index to the fuzzy sets associated to the corresponding linguistic variable. As an example, the fuzzy rule base could be represented as a matrix where each cell corresponds to the intersection of the input fuzzy sets, containing the index of the output fuzzy set associated to the rule. It is also possible to represent the fuzzy rule base as a decision table or simply as a list of rules. In these last two cases, the representation can have variable length, allowing to represent fuzzy rule sets with variable size.

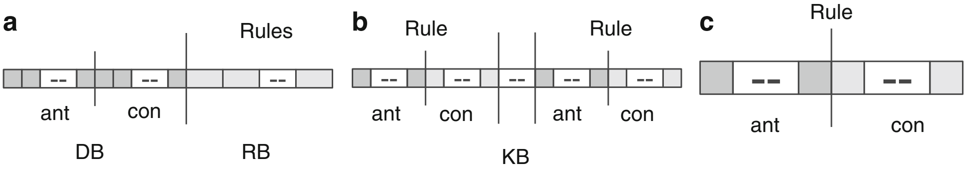
The fitness calculation depends on the selected approximation. On a Pittsburgh approximation, the fitness corresponds to the evaluation of the complete fuzzy system on the corresponding problem. It is also possible to include in the fitness calculation other factors, such as for example, penalization for fuzzy rule bases that contains many rules or fuzzy rules with superposed application areas, etc. On a Michigan or Iterative model, the fitness indicates the degree of adequacy of the rule measured independently, considering also in the Michigan model its degree of cooperation with the other rules in the population.

The definition of the variational operators depends of course on the selected approximation. If the representation allows it, standard operators of crossover and mutation can be used. However, it can be convenient (or necessary) to define specific operators. As an example, variational operators can consider factors such as the time period since the rule has been used for the last time, its overall contribution to the final result, its performance when evaluated on the set of examples, etc.

Optimization and Learning of the Complete Knowledge Base

This case is a combination of the two models described before. The knowledge base KB_i contains the initial approximation to the definition of the antecedents and consequents, and the initial approximation to the fuzzy rule base as provided by the expert. Note that KB_i can also be empty if it is expected that the algorithm must generate all the parameters of the fuzzy system by itself.

The representation of the individuals contains all the parameters that define a knowledge base in order to allow its learning or optimization. The three most used representation schemes are shown in Fig. 4. In the first scheme, each individual contains the representation of all fuzzy sets, and the representation of all fuzzy rules using indexes to refer to the corresponding fuzzy sets. In the second scheme, each individual is structured as a set of rules, where each one specifies its own input and output fuzzy sets by directly including the parameters that define them. The representation (a) is adequate for descriptive fuzzy systems, since the rules contain references



Evolutionary Fuzzy Systems, Fig. 4 Representations for the complete knowledge base adequate for (**a**) descriptive and (**b**) approximative fuzzy systems in the Pittsburgh approximation, and (**c**) representation of a single independent rule adequate for Michigan and Iterative approximations to the fuzzy sets used in their definition and can be shared by all of them. The representation (b) is adequate for approximative fuzzy systems, where each rule defines its own fuzzy sets. These two representations are adequate for the Pittsburgh approximation, while the third one (c) is adequate for the Michigan and the Iterative approximation. Of course, there can be many variations of this representations. For example, the input space partition can be predefined or obtained through fuzzy clustering algorithms, and if this partition is not expected to go under optimization, then it is not necessary to include the parameters of the input fuzzy sets in the representation.

Since this model is a combination of the two previous models, everything that was mentioned before concerning the fitness function and the variational operators also applies in this context. However, the fact that all parameters of the knowledge base are included in the representation allows to define more powerful variational operators. As an example, it is possible to define operators that decide the creation of new fuzzy sets, the elimination of some of them, and at the same time, the adaptation of the associated fuzzy rules, when for example, it is detected that there are areas in the input space that are not well covered, many rules with superimposed areas, etc. It is also possible to apply genetic programming techniques (Pedrycz 2003), which are usually used to modify the structure of the fuzzy system, adding, removing, or combining sections of the fuzzy system with the objective of generating the most adequate structure.

Final Remarks

Clearly, the integration of fuzzy systems with evolutionary algorithms allows to overcome the limitations of each model considered independently, obtaining a powerful hybrid approach, which allows to learn and optimize fuzzy systems based on expert knowledge. Previous sections have discussed in general terms the evolutionary learning model. However, in order to get more details about particular implementations, it is recommended to read the publications referenced in the next section. The presentation from Karr and Gentry (1993) is interesting, not only because it provides a nice introduction and application of evolutionary fuzzy systems, but it has the additional value of being one of the first publications in the area. The presentation of Hoffmann (2001) is an excellent introduction to evolutionary fuzzy systems used for control applications. The other publications present details on evolutionary fuzzy systems (Babuska 1998; Bonarini 1996; Cordon et al. 2001; Juang et al. 2000; Lee and Takagi 1993), including representations based on neural networks (Hoffmann 2001; Karr and Gentry 1993), evolution strategies (Alpaydtn et al. 2002), genetic programming (Pedrycz 2003) and applications of evolutionary fuzzy systems to the domain of recurrent fuzzy systems (Kavka et al. 2005). The paper by Cordon et al. (2004) provides a very comprehensive reference list about the main developments on evolutionary fuzzy systems.

It should be stressed that a very important aspect to consider in the definition of evolutionary fuzzy systems is the interpretability of the resulting fuzzy systems (Casillas et al. 2003). Even if it has been mentioned that it is possible to design an evolutionary fuzzy system without expert information, by allowing the evolutionary algorithm to define all the components of the knowledge base by itself, it must always be considered that the interpretability of the results is essential. Designing a system that solves the problem, but that works as a black box, can be adequate in other contexts, but it is not desirable at all in the context of evolutionary fuzzy systems. An evolutionary fuzzy system algorithm must provide the means so that the expert knowledge defined in fuzzy terms can be considered and used appropriately during the evolution, and also, it must guarantee an adequate interpretability degree of the resulting fuzzy system.

Recommended Reading

Alpaydtn G, Dundar G, Balktr S (2002) Evolution-based design of neural fuzzy networks using self-adapting genetic parameters. IEEE Trans Fuzzy Syst 10(2):211–221

Babuska R (1998) Fuzzy modeling for control. Kluwer Academic Press, Norwell

Bonarini A (1996) Evolutionary learning of fuzzy rules: competition and cooperation. In: Pedrycz W (ed) Fuzzy modeling: paradigms and practice. Kluwer Academic Press, Norwell

Casillas J, Cordon O, Herrera F, Magdalena L (eds) (2003) Interpretability issues in fuzzy modeling. Studies in fuzziness and soft computing, vol 128. Springer, Berlin/New York

Cordon O, Gomide F, Herrera F, Hoffmann F, Magdalena L (2004) Ten years of genetic fuzzy systems: current framework and new trends. Fuzzy Sets Syst 141:5–31

Cordon O, Herrera F, Hoffmann F (2001) Genetic fuzzy systems. World Scientific Publishing, Singapore

Hoffmann F (2001) Evolutionary algorithms for fuzzy control system design. Proc IEEE 89(9):1318–1333

Juang CF, Lin JY, Lin CT (2000) Genetic reinforcement learning through symbiotic evolution for fuzzy controller design. IEEE Trans Syst Man Cybern 30(2):290–302

Karr CL, Gentry EJ (1993) Fuzzy control of PH using genetic algorithms. IEEE Trans Fuzzy Syst 1(1): 46–53

Kavka C, Roggero P, Schoenauer M (2005) Evolution of Voronoi based fuzzy recurrent controllers. In: Proceedings of GECCO. ACM Press, NeW York, pp 1385–1392

Lee M, Takagi H (1993) Integrating design stages of fuzzy systems using genetic algorithms. In: Proceedings of the second IEEE international conference on fuzzy systems, San Francisco, pp 612–617

Pedrycz W (2003) Evolutionary fuzzy modeling. IEEE Trans Fuzzy Syst 11(5):652–665

Zadeh L (1988) Fuzzy logic. IEEE Comput 21(4): 83–93

Evolutionary Games

Moshe Sipper
Ben-Gurion University, Beer-Sheva, Israel

Definition

Evolutionary algorithms are a family of algorithms inspired by the workings of evolution by natural selection, whose basic structure is to:

1. Produce an initial *population* of individuals, these latter being candidate solutions to the problem at hand.
2. Evaluate the *fitness* of each individual in accordance with the problem whose solution is sought.
3. *While* termination condition not met *do*:
   (a) *Select* fitter individuals for reproduction
   (b) *Recombine* (*crossover*) individuals
   (c) *Mutate* individuals
   (d) *Evaluate* fitness of modified individuals
4. *End while*

Evolutionary games is the application of evolutionary algorithms to the evolution of game-playing strategies for various games, including chess, backgammon, and Robocode.

Motivation and Background

Ever since the dawn of artificial intelligence in the 1950s, games have been part and parcel of this lively field. In 1957, a year after the Dartmouth Conference that marked the official birth of AI, Alex Bernstein designed a program for the IBM 704 that played two amateur games of chess. In 1958, Allen Newell, J.C. Shaw, and Herbert Simon introduced a more sophisticated chess program (beaten in 35 moves by a 10-year-old beginner in its last official game played in 1960). Arthur L. Samuel of IBM spent much of the 1950s working on game-playing AI programs, and by 1961, he had a checkers program that could play at the master's level. In 1961 and 1963, Donald Michie described a simple trial-and-error learning system for learning how to play tic-tac-toe (or Noughts and Crosses) called MENACE (for Matchbox Educable Noughts and Crosses Engine). These are but examples of highly popular games that have been treated by AI researchers since the field's inception.

Why study games? This question was answered by Susan L. Epstein, who wrote:

> There are two principal reasons to continue to do research on games... First, human fascination with game playing is long-standing and pervasive. Anthropologists have cataloged popular games in almost every culture... Games intrigue us because they address important cognitive functions... The second reason to continue game-playing research

> is that some difficult games remain to be won, games that people play very well but computers do not. These games clarify what our current approach lacks. They set challenges for us to meet, and they promise ample rewards (Epstein 1999).

Studying games may thus advance our knowledge in both cognition and artificial intelligence, and, last but not least, games possess a competitive angle which coincides with our human nature, thus motivating both researcher and student alike.

Even more strongly, Laird and van Lent proclaimed that:

> . . . interactive computer games are the killer application for human-level AI. They are the application that will soon need human-level AI, and they can provide the environments for research on the right kinds of problems that lead to the type of the incremental and integrative research needed to achieve human-level AI (Laird and van Lent 2000).

Recently, evolutionary algorithms have proven a powerful tool that can automatically "design" successful game-playing strategies for complex games (Azaria and Sipper 2005a,b; Hauptman and Sipper 2005b, 2007a,b; Shichel et al. 2005; Sipper et al. 2007).

Structure of the Learning System

Genetic Programming

Genetic programming is a subclass of evolutionary algorithms, wherein a *population* of individual programs is evolved, each program comprising *functions* and *terminals*. The functions are usually arithmetic and logic operators that receive a number of arguments as input and compute a result as output; the terminals are zero-argument functions that serve both as constants and as sensors, the latter being a special type of function that queries the domain environment.

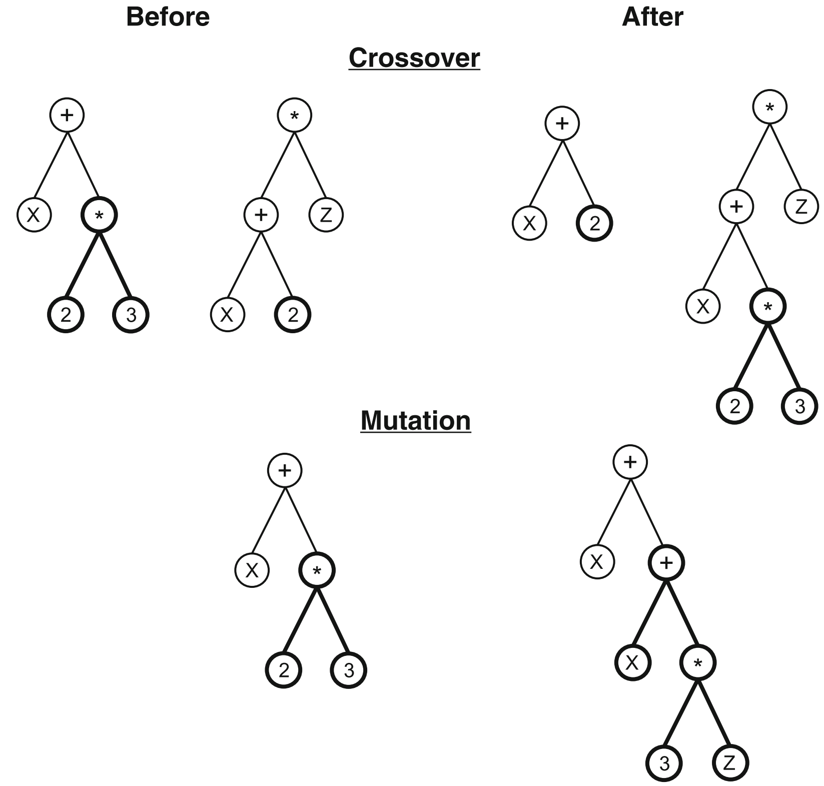
The main mechanism behind genetic programming is precisely that of a generic evolutionary algorithm (Sipper 2002; Tettamanzi and Tomassini 2001), namely, the repeated cycling through four operations applied to the entire population: evaluate-select-crossover-mutate. Starting with an initial population of randomly generated programs, each individual is evaluated in the domain environment and assigned a *fitness* value representing how well the individual solves the problem at hand. Being randomly generated, the first-generation individuals usually exhibit poor performance. However, some individuals are better than others, that is, (as in nature) variability exists, and through the mechanism of natural (or, in our case, artificial) selection, these have a higher probability of being selected to parent the next generation. The size of the population is finite and usually constant.

Specifically, first a genetic operator is chosen at random; then, depending on the operator, one or two individuals are selected from the current population using a *selection operator*, one example of which is *tournament selection*: Randomly choose a small subset of individuals, and then select the one with the best fitness. After the probabilistic selection of better individuals, the chosen genetic operator is used to construct the next generation. The most common operators are:

- Reproduction (unary): Copy one individual to the next generation with no modifications. The main purpose of this operator is to preserve a small number of good individuals.
- Crossover (binary): Randomly select an internal node in each of the two individuals and swap the subtrees rooted at these nodes. An example is shown in Fig. 1.
- Mutation (unary): Randomly select a node from the tree, delete the subtree rooted at that node, and then "grow" a new subtree in its stead. An example is shown in Fig. 1 (the growth operator as well as crossover and mutation are described in detail in Koza 1992).

The generic genetic programming flowchart is shown in Fig. 2. When one wishes to employ genetic programming, one needs to define the following six desiderata:

1. Program architecture
2. Set of terminals



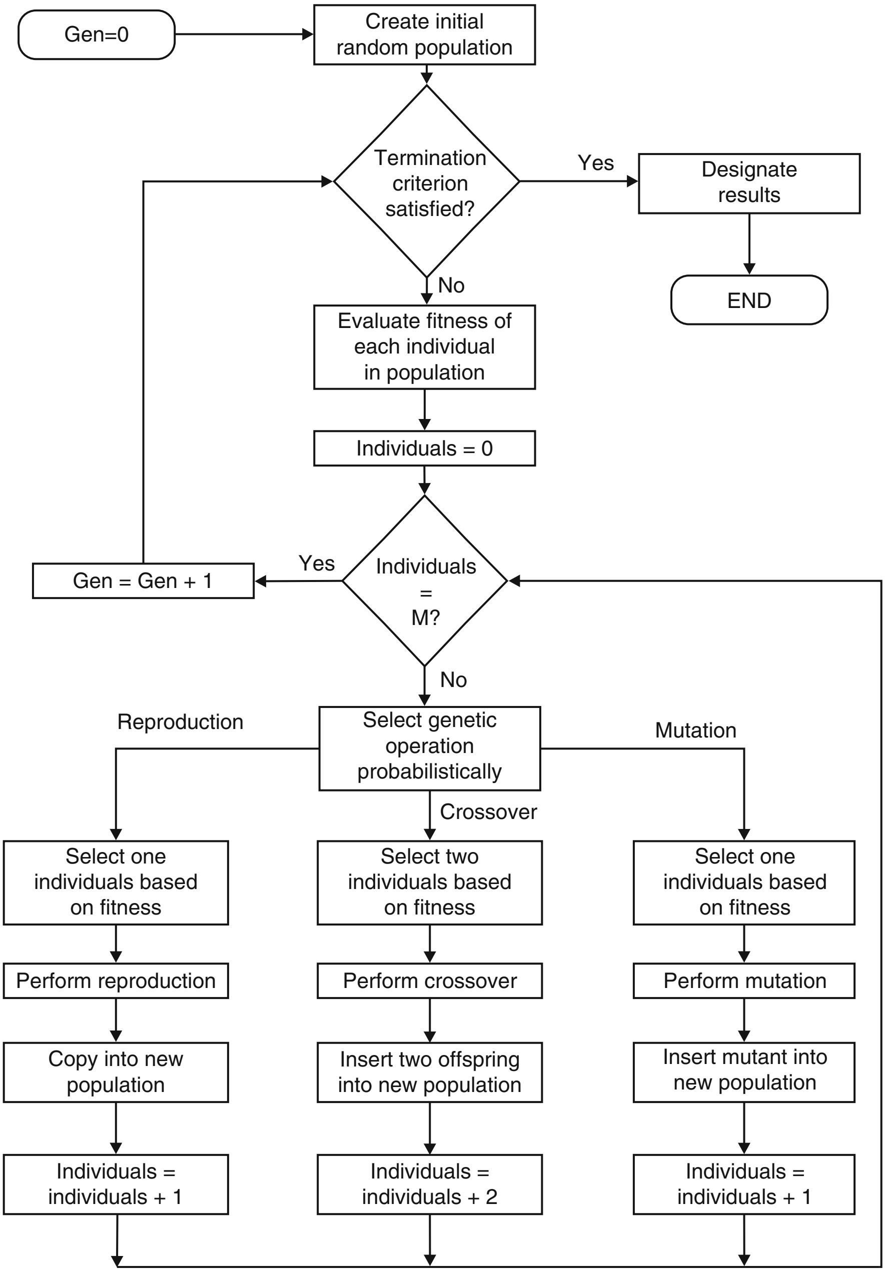
Evolutionary Games, Fig. 1 Genetic operators in genetic programming. LISP programs are depicted as trees. Crossover (*top*): Two subtrees (marked in *bold*) are selected from the parents and swapped. Mutation (*bottom*): A subtree (marked in *bold*) is selected from the parent individual and removed. A new subtree is grown instead

3. Set of functions
4. Fitness measure
5. Control parameters
6. Manner of designating result and terminating run

Evolving Game-Playing Strategies

Recently, we have shown that complex and successful game-playing strategies can be attained via genetic programming. We focused on three games (Azaria and Sipper 2005a,b; Hauptman and Sipper 2005b, 2007a,b; Shichel et al. 2005; Sipper et al. 2007):

1. *Backgammon.* Evolves a full-fledged player for the nondoubling-cube version of the game (Azaria and Sipper 2005a,b; Sipper et al. 2007).
2. *Chess* (endgames). Evolves a player able to play endgames (Hauptman and Sipper 2005b, 2007a,b; Sipper et al. 2007). While endgames typically contain but a few pieces, the problem of evaluation is still hard, as the pieces are usually free to move all over the board, resulting in complex game trees – both deep and with high branching factors. Indeed, in the chess lore, much has been said and written about endgames.
3. *Robocode.* A simulation-based game in which robotic tanks fight to destruction in a closed arena (robocode.alphaworks.ibm.com). The programmers implement their robots in the Java programming language and can test their creations either by using a graphical environment in which battles are held or by submitting them to a central Web site where


Gen=0
Create initial random population
Termination criterion satisfied?
Yes
Designate results
END
No
Evaluate fitness of each individual in population
Individuals = 0
Individuals = M?
Yes
Gen = Gen + 1
No
Select genetic operation probabilistically
Reproduction
Crossover
Mutation
Select one individuals based on fitness
Select two individuals based on fitness
Select one individuals based on fitness
Perform reproduction
Perform crossover
Perform mutation
Copy into new population
Insert two offspring into new population
Insert mutant into new population
Individuals = individuals + 1
Individuals = individuals + 2
Individuals = individuals + 1

Evolutionary Games, Fig. 2 Generic genetic programming flowchart (based on Koza 1992). M is the population size, and Gen is the generation counter. The termination criterion can be the completion of a fixed number of generations or the discovery of a good-enough individual

online tournaments regularly take place. Our goal here has been to evolve Robocode players able to rank high in the international league (Shichel et al. 2005; Sipper et al. 2007).

A strategy for a given player in a game is a way of specifying which choice the player is to make at every point in the game from the set of allowable choices at that point, given all the information that is available to the player at that point (Koza 1992). The problem of discovering a strategy for playing a game can be viewed as one of seeking a computer program. Depending on the game, the program might take as input the entire history of past moves or just the current state of the game. The desired program then produces the next move as output. For some games, one might evolve a complete strategy that addresses every situation tackled. This proved to work well with Robocode, which is a dynamic game, with relatively few parameters and little need for past history.

In a two-player game, such as chess or backgammon, players move in turn, each trying to win against the opponent according to specific rules (Hong et al. 2001). The course of the game may be modeled using a structure known as an adversarial game tree (or simply game tree), in which nodes are the positions in the game and edges are the moves. By convention, the two players are denoted as MAX and MIN, where MAX is the player who moves first. Thus, all nodes at odd-numbered tree levels are game positions where MAX moves next (labeled MAX nodes). Similarly, nodes on even levels are called MIN nodes and represent positions in which MIN (opponent) moves next.

The complete game tree for a given game is the tree starting at the initial position (the root) and containing all possible moves (edges) from each position. *Terminal nodes* represent positions where the rules of the game determine whether the result is a win, a draw, or a loss. Although the game tree for the initial position is an explicit representation of all possible paths of the game, therefore theoretically containing all the information needed to play perfectly, for most (nontrivial) games, it is extremely large, and constructing it is not feasible. For example, the complete chess game tree consists of roughly 10^{43} nodes (Shannon 1950).

When the game tree is too large to be generated completely, only a partial tree (called a search tree) is generated instead. This is accomplished by invoking a *search algorithm*, deciding which nodes are to be developed at any given time and when to terminate the search (typically at nonterminal nodes due to time constraints). During the search, some nodes are evaluated by means of an *evaluation function* according to given heuristics. This is done mostly at the leaves of the tree. Furthermore, search can start from any position and not just at the beginning of the game.

Because we are searching for a winning strategy, we need to find a good next move for the current player, such that no matter what the opponent does thereafter, the player's chances of winning the game are as high as possible. A well-known method called the *minimax* search (Campbell and Marsland 1983; Kaindl 1988) has traditionally been used, and it forms the basis for most methods still in use today. This algorithm performs a depth-first search (the depth is usually predetermined), applying the evaluation function to the leaves of the tree and propagating these values upward according to the minimax principal: at MAX nodes, select the maximal value and at MIN nodes – the minimal value. The value is ultimately propagated to the position from which the search had started.

With games such as backgammon and chess, one can couple a current-state evaluator (e.g., board evaluator) with a next-move generator. One can then go on to create a minimax tree, which consists of all possible moves, counter moves, counter counter-moves, and so on; for real-life games, such a tree's size quickly becomes prohibitive. The approach we used with backgammon and chess is to derive a very shallow, single-level tree and evolve "smart" evaluation functions. Our artificial player is thus created by combining an evolved board evaluator with a simple program that generates all next-move boards (such programs can easily be written for backgammon and chess).

In what follows, we describe the definition of the six items necessary in order to employ genetic programming, as delineated in the previous section: program architecture, set of terminals, set of functions, fitness measure, control parameters, and manner of designating result and terminating run. Due to lack of space, we shall elaborate upon one game – Robocode – and only summarize the major results for backgammon and chess.

Example: Robocode

Program Architecture

A Robocode player is written as an event-driven Java program. A main loop controls the tank activities, which can be interrupted on various occasions, called *events*. The program is limited to four lines of code, as we were aiming for the HaikuBot category, one of the divisions of the international league with a four-line code limit. The main loop contains one line of code that directs the robot to start turning the gun (and the mounted radar) to the right. This insures that within the first gun cycle, an enemy tank will be spotted by the radar, triggering a *ScannedRobotEvent*. Within the code for this event, three additional lines of code were added, each controlling a single actuator and using a single numerical input that was supplied by a genetic programming-evolved subprogram. The first line instructs the tank to move to a distance specified by the first evolved argument. The second line instructs the tank to turn to an azimuth specified by the second evolved argument. The third line instructs the gun (and radar) to turn to an azimuth specified by the third evolved argument (Fig. 3).

Terminal and Function Sets

We divided the terminals into three groups according to their functionality (Shichel et al. 2005) (Fig. 4):

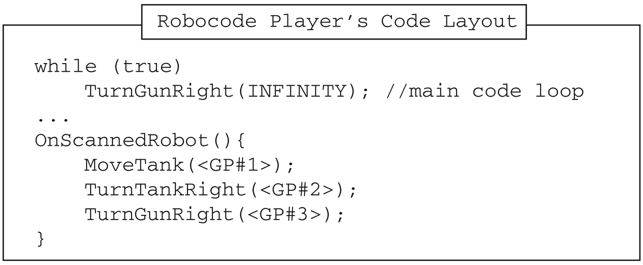
1. Game-status indicators: A set of terminals that provide real-time information on the game status, such as last enemy azimuth, current tank position, and energy levels.
2. Numerical constants: Two terminals, one providing the constant 0 and the other being an ephemeral random constant (ERC). This latter terminal is initialized to a random real numerical value in the range [−1, 1] and does not change during evolution.
3. Fire command: This special function is used to curtail one line of code by not implementing the fire actuator in a dedicated line.

Fitness Measure

We explored two different modes of learning: using a fixed external opponent as teacher and coevolution – letting the individuals play against each other; the former proved better. However, not just one but three external opponents were used to measure performance; these adversaries were downloaded from the HaikuBot league (robocode.yajags.com). The fitness value of an individual equals its average fractional score (over three battles).

Control Parameters and Run Termination

The major evolutionary parameters (Koza 1992) were population size (256), generation count (between 100 and 200), selection method (tournament), reproduction probability (0), crossover probability (0.95), and mutation probability (0.05). An evolutionary run terminates when fitness is observed to level off. Since the game is highly nondeterministic, a "lucky" individual might attain a higher fitness value than better overall individuals. In order to obtain a more accurate measure for the evolved players, we let each of them do battle for 100 rounds against 12 different adversaries (one at a time). The results were used to extract the



Robocode Player's Code Layout

```
while (true)
    TurnGunRight(INFINITY); //main code loop
...
OnScannedRobot(){
    MoveTank(<GP#1>);
    TurnTankRight(<GP#2>);
    TurnGunRight(<GP#3>);
}
```

Evolutionary Games, Fig. 3 Robocode player's code layout (HaikuBot division)

a

Energy()	Returns the remaining energy of the player
Heading()	Returns the current heading of the player
X()	Returns the current horizontal position of the player
Y()	Returns the current vertical position of the player
MaxX()	Returns the horizontal battlefield dimension
MaxY()	Returns the vertical battlefield dimension
EnemyBearing()	Returns the current enemy bearing, relative to the current player's heading
EnemyDistance()	Returns the current distance to the enemy
EnemyVelocity()	Returns the current enemy's velocity
EnemyHeading()	Returns the current enemy heading, relative to the current player's heading
EnemyEnergy()	Returns the remaining energy of the enemy
Constant()	An ERC (Ephemeral Random Constant) in the range [–1,1]
Random()	Returns a random real number in the range [–1,1]
Zero()	Returns the constant 0

b

Add(F, F)	Add two real numbers
Sub(F, F)	Subtract two real numbers
Mul(F, F)	Multiply two real numbers
Div(F, F)	Divide first argument by second, if denominator non-zero, otherwise return zero
Abs(F)	Absolute value
Neg(F)	Negative value
Sin(F)	Sine function
Cos(F)	Cosine function
ArcSin(F)	Arcsine function
ArcCos(F)	Arccosine function
IfGreater(F, F, F, F)	If first argument greater than second, return value of third argument, else return value of fourth argument
IfPositive(F, F, F)	If first argument is positive, return value of second argument, else return value of third argument
Fire(F)	If argument is positive, execute fire command with argument as firepower and return 1; otherwise, do nothing and return 0

Evolutionary Games, Fig. 4 Robocode representation. (**a**) Terminal set (**b**) Function set (*F*: Float)

top player – to be submitted to the international league.

Results

We submitted our top player to the HaikuBot division of the international league. At its very first tournament, it came in third, later climbing to first place of 28 (robocode.yajags.com/20050625/haiku-1v1.html). All other 27 programs, defeated by our evolved strategy, were written by humans. For more details on GP-Robocode see Shichel et al. (2005) and Sipper et al. (2007).

Backgammon and Chess: Major Results

Backgammon

We pitted our top evolved backgammon players against *Pubeval*, a free, public-domain board evaluation function written by Tesauro. The program – which plays well – has become the de facto yardstick used by the growing commu-

Evolutionary Games, Table 1 Percent of wins, advantages, and draws for the best GP-EndChess player in the tournament against two top competitors

	%Wins	%Advs	%Draws
Master	6.00	2.00	68.00
CRAFTY	2.00	4.00	72.00

nity of backgammon-playing program developers. Our top evolved player was able to attain a win percentage of 62.4 % in a tournament against Pubeval, about 10 % higher (!) than the previous top method. Moreover, several evolved strategies were able to surpass the 60 % mark, and most of them outdid all previous works. For more details on GP-Gammon, see Azaria and Sipper (2005a) and Sipper et al. (2007).

Chess (Endgames)

We pitted our top evolved chess-endgame players against two very strong external opponents: (1) a program we wrote ("Master") based upon consultation with several high-ranking chess players (the highest being Boris Gutkin, ELO 2400, International Master) and (2) CRAFTY – a world-class chess program, which finished second in the 2004 World Computer Speed Chess Championship (www.cs.biu.ac.il/games/). Speed chess ("blitz") involves a time limit per move, which we imposed both on CRAFTY and on our players. Not only did we thus seek to evolve good players, but ones who play well *and fast*. Results are shown in Table 1. As can be seen, GP-EndChess manages to hold its own, and even wins, against these top players. For more details on GP-EndChess, see Sipper et al. (2007) and Hauptman and Sipper (2005b).

Deeper analysis of the strategies developed (Hauptman and Sipper 2005a) revealed several important shortcomings, most of which stemmed from the fact that they used deep knowledge and little search (typically, they developed only *one* level of the search tree). Simply increasing the search depth would not solve the problem, since the evolved programs examine each board very thoroughly, and scanning many boards would increase time requirements prohibitively. And so we turned to evolution to find an optimal way to overcome this problem: How to add more search at the expense of less knowledgeable (and thus less time-consuming) node evaluators, while attaining better performance. In Hauptman and Sipper (2007b) *we evolved the search algorithm itself*, focusing on the *Mate-In-N* problem: find a key move such that even with the best possible counterplays, the opponent cannot avoid being mated in (or before) move N. We showed that our evolved search algorithms successfully solve several instances of the Mate-In-N problem, for the hardest ones developing 47 % less game-tree nodes than CRAFTY. Improvement is thus not over the basic alpha-beta algorithm, but over a world-class program using all standard enhancements (Hauptman and Sipper 2007b).

Finally, in Hauptman and Sipper (2007a), we examined a strong evolved chess-endgame player, focusing on the player's emergent capabilities and tactics in the context of a chess match. Using a number of methods, we analyzed the evolved player's building blocks and their effect on play level. We concluded that evolution has found combinations of building blocks that are far from trivial and cannot be explained through simple combination – thereby indicating the possible emergence of complex strategies.

Cross-References

- ▶ Evolutionary Algorithms
- ▶ Evolutionary Computation
- ▶ Evolutionary Computing
- ▶ Genetic Programming

Recommended Reading

Azaria Y, Sipper M (2005a) GP-Gammon: genetically programming backgammon players. Genet Program Evolvable Mach 6(3):283–300

Azaria Y, Sipper M (2005b) GP-Gammon: using genetic programming to evolve backgammon players.

In: Keijzer M, Tettamanzi A, Collet P, van Hemert J, Tomassini M (eds) Proceedings of 8th European conference on genetic programming (EuroGP2005), Lausanne. LNCS, vol 3447. Springer, Heidelberg, pp 132–142

Campbell MS, Marsland TA (1983) A comparison of minimax tree search algorithms. Artif Intell 20:347–367

Epstein SL (1999) Game playing: the next moves. In: Proceedings of the sixteenth national conference on artificial intelligence, Orland. AAAI, Menlo Park, pp 987–993

Hauptman A, Sipper M (2005a) Analyzing the intelligence of a genetically programmed chess player. In: Late breaking papers at the 2005 genetic and evolutionary computation conference (GECCO 2005), Washington, DC

Hauptman A, Sipper M (2005b) GP-EndChess: using genetic programming to evolve chess endgame players. In: Keijzer M, Tettamanzi A, Collet P, van Hemert J, Tomassini M (eds) Proceedings of 8th European conference on genetic programming (EuroGP2005), Lausanne. LNCS, vol 3447. Springer, Heidelberg, pp 120–131

Hauptman A, Sipper M (2007a) Emergence of complex strategies in the evolution of chess endgame players. Adv Complex Syst 10(Suppl 1):35–59

Hauptman A, Sipper M (2007b) Evolution of an efficient search algorithm for the mate-in-N problem in chess. In: Ebner M, O'Neill M, Ekárt A, Vanneschi L, Esparcia-Alcázar AI (eds) Proceedings of 10th European conference on genetic programming (EuroGP2007), Valencia. LNCS, vol 4445. Springer, Heidelberg, pp 78–89

Hong T-P, Huang K-Y, Lin W-Y (2001) Adversarial search by evolutionary computation. Evol Comput 9(3):371–385

Kaindl H (1988) Minimaxing: theory and practice. AI-Mag 9(3):69–76

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT, Cambridge

Laird JE, van Lent M (2000) Human-level AI's killer application: interactive computer games. In: AAAI-00: proceedings of the 17th national conference on artificial intelligence, Austin. MIT, Cambridge, pp 1171–1178

Shannon CE (1950) Automatic chess player. Sci Am 48:182

Shichel Y, Ziserman E, Sipper M (2005) GP-Robocode: using genetic programming to evolve robocode players. In: Keijzer M, Tettamanzi A, Collet P, van Hemert J, Tomassini M (eds) Proceedings of 8th European conference on genetic programming (EuroGP2005), Lausanne. LNCS, vol 3447. Springer, Heidelberg, pp 143–154

Sipper M (2002) Machine nature: the coming age of bio-inspired computing. McGraw-Hill, New York

Sipper M, Azaria Y, Hauptman A, Shichel Y (2007) Designing an evolutionary strategizing machine for game playing and beyond. IEEE Trans Syst Man Cybern Part C Appl Rev 37(4):583–593

Tettamanzi A, Tomassini M (2001) Soft computing: integrating evolutionary, neural, and fuzzy systems. Springer, Berlin

Evolutionary Grouping

▶ Evolutionary Clustering

Evolutionary Kernel Learning

Christian Igel
Department of Computer Science, University of Copenhagen, Copenhagen, Denmark

Definition

Evolutionary kernel learning stands for using ▶ evolutionary algorithms to design the ▶ kernel function for a ▶ kernel method.

Motivation and Background

In kernel-based learning algorithms, the kernel function implicitly defines the feature space in which the algorithm operates. The kernel determines the scalar product and thereby the metric in the feature space. Choosing the right kernel function is crucial for the training accuracy and generalization performance of the learning machine. The choice may also influence the runtime and storage complexity during and after training.

The kernel is usually not adapted by the kernel method itself; choosing it is a ▶ model selection problem. In practice, the kernel function is selected from an a priori fixed class. When a parameterized family of kernel functions is considered, kernel adaptation reduces to finding appropriate parameters. The most frequently used method to determine these values is grid search. In simple grid search, the parameters are varied with a fixed step-size through a range of values, and the performance of each combination is measured. Because of its computational complexity, grid search is only suitable for the adjustment of a few parameters. Furthermore, the choice of the discretization of the search space may be crucial. Gradient-based approaches are perhaps the most elaborate techniques for adapting real-valued kernel parameters, see the articles by Chapelle et al. (2002) and Glasmachers and Igel (2005) and references therein. To use these methods, however, the class of kernel functions must have a differentiable structure. Furthermore, score functions for assessing the parameter performance that are not differentiable and/or piecewise constant may cause problems. Evolutionary kernel learning does not suffer from these limitations. Additionally, it allows for ▶ multi-objective optimization (MOO) to address several kernel design criteria.

Structure of Learning System

Canonical evolutionary kernel learning can be described as an evolutionary algorithm (EA) in which the individuals encode kernel functions, see Fig. 1. These individuals are evaluated by determining the task-specific performance of the kernel they represent. Two special aspects must be considered when designing an EA for kernel learning. First, one must decide how to assess the performance (i.e., the fitness) of a particular kernel. That is, model selection criteria have to be defined depending on the problem at hand. Second, one must also specify the subset of possible kernel functions to be searched by the EA. This leads to the questions of how to encode the kernels and which variation operators to employ.

Assessing Fitness: Model Selection Criteria

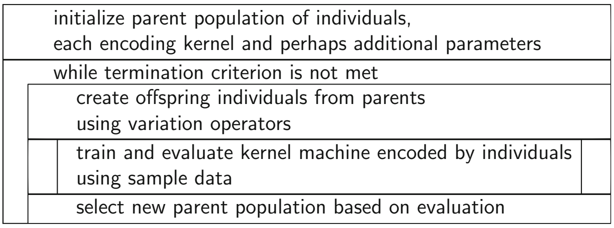
The following presents some performance indices that have been considered for evolutionary kernel learning. They can be used individually or in linear combination for single-objective optimization. In MOO several of these criteria can be used as different objectives.

It is important to note that, although many of these measures are designed to improve ▶ generalization, kernel learning can lead to ▶ overfitting if only limited data are used in the model selection process (e.g., if in every generation, the same small data sets are used to assess performance). Regularization (e.g., in a Bayesian framework) can be used to prevent overfitting. If enough data are available, it is advisable to monitor the generalization behavior of kernel learning using independent data. For example, external data can be used for the early stopping of evolutionary kernel learning (Igel 2013).

Accuracy on Sample Data

The most straightforward way to evaluate a model is to consider its performance on sample data. The empirical risk given by the error on the training data can be considered, but it does not measure generalization. To estimate the generalization performance, the accuracy on data not used for training is evaluated. In the simplest case, the available data are split into a training and validation set, with the first used for learning and the second for subsequent performance assessment. A theoretically sound and simple method is ▶ cross-validation (CV). Cross-validation makes better use of the available data, but it is more computationally demanding.

Using holdout validation sets alone does not prevent overfitting if the validation sets are small and are reused in every generation. If sufficient data are available, it is advisable to resample the data used for fitness evaluation in each generation to prevent overfitting (Igel 2013).



```
initialize parent population of individuals,
each encoding kernel and perhaps additional parameters
while termination criterion is not met
    create offspring individuals from parents
    using variation operators
    train and evaluate kernel machine encoded by individuals
    using sample data
    select new parent population based on evaluation
```

Evolutionary Kernel Learning, Fig. 1 Canonical evolutionary kernel learning algorithm

If ▶ classification is considered, it may be reasonable to split the classification error into false negative and false positive rates and to view ▶ sensitivity and ▶ specificity as two separate objectives (Suttorp and Igel 2006).

Measures Derived from Bounds on the Generalization Performance

Statistical learning theory provides estimates of and bounds on the expected generalization error of learning machines. These results can be utilized as criteria for model selection. One has to keep in mind that the model selection process may lead to violations of the assumptions underlying the corresponding theorems from statistical learning theory, in which case the computed performance indicators can not be strictly interpreted as bounds and unbiased estimates.

An example drawing inspiration from radius-margin bounds for evolving kernels for ▶ support vector machines (SVMs) for classification is given by Igel (2005). Furthermore, the number of support vectors (SVs) was optimized in combination with the empirical risk (Igel 2005). For hard-margin SVMs, the fraction of SVs is an upper bound on the leave-one-out error (e.g., Chapelle et al. 2002).

Number of Input Variables

Variable selection refers to the ▶ feature selection problem of choosing input variables that are best suited for the learning task. Masking a subset of variables can be viewed as modifying the kernel. Considering only a subset of feature dimensions decreases the computational complexity of the learning machine. When deteriorating feature dimensions are removed, the overall performance may increase. Reducing the number of input variables is therefore a common objective, which can be achieved by using single-objective (Eads et al. 2002; Fröhlich et al. 2004; Jong et al. 2004; Miller et al. 2003) or multi-objective (Pang and Kasabov 2004; Shi et al. 2004) evolutionary kernel learning.

Space and Time Complexity of the Classifier

Sometimes it can be very important to have fast kernel methods (e.g., for meeting real-time constraints). Thus, the execution time may be considered in the performance assessment during evolutionary kernel learning.

Reducing the number of input variables speeds up kernel methods. The space and time complexity of SVMs also scales with the number of SVs. This is an additional reason to consider minimization of the number of SVs as an objective in evolutionary model selection for SVMs (Igel 2005; Suttorp and Igel 2006).

Multi-objective Optimization

The design of a learning machine can be considered as a MOO problem. For example, accuracy and complexity can be viewed as different, and probably conflicting, objectives. The goal of MOO is to approximate a diverse set of Pareto-optimal solutions (i.e., solutions that cannot be improved in one objective without getting worse in another one), which provide insights into the trade-offs between the objectives. Evolutionary multi-objective algorithms have become popular for MOO. Applications of multi-objective evolutionary kernel learning combining some of the performance measures listed above can be found in the work of Igel (2005), Pang and Kasabov (2004), Shi et al. (2004), and Suttorp and Igel (2006).

Coevolution

▶ Coevolutionary learning also finds application in evolutionary kernel design. For instance, Gagné et al. (2006) suggest coevolution to speed up the evaluation and optimization of kernel nearest neighbor classifiers. They evolve three different species. The first encodes the kernels, the second a subset of the training examples used for building the classifier, and the third a subset of examples used for fitness evaluation. Kernels and training examples cooperate, while the third species competes with the kernels.

Encoding and Variation Operators

The sheer complexity of the space of possible kernel functions makes it necessary to restrict the search to a particular class of kernel functions. This restriction essentially determines the representation and the operators used in evolutionary kernel learning.

When a parameterized family of mappings is considered, the kernel parameters can be encoded more or less directly in a real-valued EA. This is a frequently used representation, for example, for Gaussian kernel functions.

For variable selection, a binary encoding can be appropriate. For choosing a subset out of d variables, bitstrings of length d can be considered, where each bit indicates whether a particular input variable is considered in the kernel computation or not (Pang and Kasabov 2004; Shi et al. 2004).

Kernels can be built from other kernels. For example, if k_1 and k_2 are kernel functions on X, then $ak_1(\boldsymbol{x}, z) + bk_2(\boldsymbol{x}, z)$ and $a \exp(-bk_1(\boldsymbol{x}, z))$ for $\boldsymbol{x}, z \in X, a, b \in \mathbb{R}^+$ are also kernels on X. This suggests a variable-length representation in which the individuals encode expressions that evaluate to kernel functions.

Given these different search spaces, it is not surprising that the aspects of all major branches of evolutionary computation have been used in evolutionary kernel learning: genetic algorithms (Fröhlich et al. 2004), genetic programming (Howley and Madden 2005; Gagné et al. 2006), evolution strategies (Igel 2005; Friedrichs and Igel 2005), and evolutionary programming (Runarsson and Sigurdsson 2004).

In general, kernel methods assume that the kernel (or at least the ▶ Gram matrix in the training process) is ▶ positive semidefinite (psd). Therefore, it is advisable to restrict the search space such that only psd functions evolve. Other ways of dealing with the problem of ensuring positive semidefiniteness are to assign lethal fitness values to individuals not encoding proper kernels or to construct a psd Gram matrix from the matrix $\boldsymbol{M}$ induced by the training data and a non-psd "kernel" function. The latter can be achieved by subtracting the smallest eigenvalue of $\boldsymbol{M}$ from its diagonal entries.

Gaussian Kernels

Gaussian kernel functions are prevalent. Their general form is $k(\boldsymbol{x}, z) = \exp\left(-(\boldsymbol{x} - z)^{\mathrm{T}} \boldsymbol{A}(\boldsymbol{x} - z)\right)$ for $\boldsymbol{x}, z \in \mathbb{R}^n$ and symmetric positive definite (pd) matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$. When adapting $\boldsymbol{A}$, the issue of ensuring that the optimization algorithm only generates pd matrices $\boldsymbol{A}$ arises. This can be achieved by an appropriate parametrization of $\boldsymbol{A}$. Often the search is restricted to matrices of the form $\gamma \boldsymbol{I}$, where $\boldsymbol{I}$ is the unit matrix and $\gamma \in \mathbb{R}^+$ is the only adjustable parameter. However, allowing more flexibility has proven to be beneficial in certain applications (e.g., see Chapelle et al. 2002; Friedrichs and Igel 2005; Glasmachers and Igel 2005). It is straightforward to consider diagonal matrices with positive elements to allow for independent scaling factors weighting the input components. However, only by dropping this restriction one can achieve invariance against both rotation and scaling of the input space. A real-valued encoding that maps onto the set of all symmetric pd matrices can be used such that all modifications of the parameters result in feasible kernels, see the articles by Friedrichs and Igel (2005), Glasmachers and Igel (2005), and Suttorp and Igel (2006) for different parametrizations.

Optimizing Additional Hyperparameters

One of the advantages of evolutionary kernel learning is that it can be easily combined with an optimization of additional hyperparameters of the kernel method. The most prominent example is to encode not only the kernel but also the regularization parameter in evolutionary model selection for SVMs.

Application Example

Notable applications of evolutionary kernel learning include the design of classifiers in bioinformatics (Mersch et al. 2007; Pang and Kasabov 2004; Shi et al. 2004). Let us consider the work by Mersch et al. (2007) as an instructive example. Here, the parameters of a sequence kernel are evolved to improve the prediction of gene starts in DNA sequences. The kernel can be viewed as a weighted sum of 64 kernels, each measuring similarity with respect to a particular trinucleotide sequence (codon). The 64 weights $w_1, \ldots, w_{64}$ are optimized together with an additional global kernel parameter σ and a regularization parameter C for the SVM. Each individual stores $\boldsymbol{x} \in \mathbb{R}^{66}$, where $(w_1, \ldots, w_{64}, \sigma, C)^{\mathrm{T}} = (\exp(x_1), \ldots, \exp(x_{64}), |x_{65}|, |x_{66}|)^{\mathrm{T}}$. An evolution strategy is applied, using additive multivariate Gaussian mutation and weighted global recombination for variation and rank-based selection. The fitness is determined by five fold cross-validation. The evolved kernels lead to higher classification rates, and the adapted weights reveal the importance of particular codons for the task at hand.

Cross-References

▸ Neuroevolution

Recommended Reading

Chapelle O, Vapnik V, Bousquet O, Mukherjee S (2002) Choosing multiple parameters for support vector machines. Mach Learn 46(1):131–159

Eads DR, Hill D, Davis S, Perkins SJ, Ma J, Porter RB et al (2002) Genetic algorithms and support vector machines for time series classification. In: Bosacchi B, Fogel DB, Bezdek JC (eds) Applications and science of neural networks, fuzzy systems, and evolutionary computation V. Proceedings of the SPIE, vol 4787. SPIE–The International Society for Optical Engineering, Bellington, pp 74–85

Friedrichs F, Igel C (2005) Evolutionary tuning of multiple SVM parameters. Neurocomputing 64(C):107–117

Fröhlich H, Chapelle O, Schölkopf B (2004) Feature selection for support vector machines using genetic algorithms. Int J Artif Intell Tools 13(4):791–800

Gagné C, Schoenauer M, Sebag M, Tomassini M (2006) Genetic programming for kernel-based learning with co-evolving subsets selection. In: Runarsson TP, Beyer H-G, Burke E, Merelo-Guervós JJ, Whitley LD, Yao X (eds) Parallel problem solving from nature (PPSN IX). LNCS, vol 4193. Springer, Berlin, pp 1008–1017

Glasmachers T, Igel C (2005) Gradient-based adaptation of general Gaussian kernels. Neural Comput 17(10):2099–2105

Howley T, Madden M (2005) The genetic kernel support vector machine: description and evaluation. Artif Intell Rev 24(3):379–395

Igel C (2005) Multi-objective model selection for support vector machines. In: Coello Coello CA, Zitzler E, Hernandez Aguirre A (eds) Proceedings of the third international conference on evolutionary multi-criterion optimization (EMO 2005). LNCS, vol 3410. Springer, Berlin, pp 534–546

Igel C (2013) A note on generalization loss when evolving adaptive pattern recognition systems. IEEE Transact Evol Comput 17(3):345–352

Jong K, Marchiori E, van der Vaart A (2004) Analysis of proteomic pattern data for cancer detection. In: Raidl GR, Cagnoni S, Branke J, Corne DW, Drechsler R, Jin Y et al (eds) Applications of evolutionary computing. LNCS, vol 3005. Springer, Berlin, pp 41–51

Mersch B, Glasmachers T, Meinicke P, Igel C (2007) Evolutionary optimization of sequence kernels for detection of bacterial gene starts. Int J Neural Syst 17(5):369–381

Miller MT, Jerebko AK, Malley JD, Summers RM (2003) Feature selection for computer-aided polyp detection using genetic algorithms. In: Clough AV, Amini AA (eds) Medical imaging 2003: physiology and function: methods, systems, and applications. Proceedings of the SPIE, Santa Clara, vol 5031, pp 102–110

Pang S, Kasabov N (2004) Inductive vs. transductive inference, global vs. local models: SVM, TSVM, and SVMT for gene expression classification problems. In: International joint conference on neural networks (IJCNN 2004), vol 2. IEEE Press, Washington, DC, pp 1197–1202

Runarsson TP, Sigurdsson S (2004) Asynchronous parallel evolutionary model selection for support vector machines. Neural Inf Process – Lett Rev 3(3):59–68

Shi SYM, Suganthan PN, Deb K (2004) Multi-class protein fold recognition using multi-objective evolutionary algorithms. In: IEEE symposium on computational intelligence in bioinformatics and computational biology. IEEE Press, Washington, DC, pp 61–66

Suttorp T, Igel C (2006) Multi-objective optimization of support vector machines. In: Jin Y (ed) Multi-objective machine learning. Studies in computational intelligence, vol 16. Springer, Berlin, pp 199–220

Evolutionary Robotics

Phil Husbands
Department of Informatics, Centre for
Computational Neuroscience and Robotics,
University of Sussex, Brighton, UK

Abstract

Evolutionary robotics uses evolutionary search methods to fully or partially design robotic systems, including their control

systems and sometimes their morphologies and sensor/actuator properties. Such methods are used in a range of ways from the fine-tuning or optimization of established designs to the creation of completely novel designs. There are many applications of evolutionary robotics from wheeled to legged to swimming to flying robots. A particularly active area is the use of evolutionary robotics to synthesize embodied models of complete agent behaviors in order to help explore and generate hypotheses in neurobiology and cognitive science.

Synonyms

Embodied evolutionary learning; Evolution of agent behaviors; Evolution of robot control

Definition

Evolutionary robotics involves the use of ▸ evolutionary computing techniques to automatically develop some or all of the following properties of a robot: the control system, the body morphology, and the sensor and motor properties and layout. Populations of artificial genomes (usually lists of characters and numbers) encode properties of autonomous mobile robots required to carry out a particular task or to exhibit some set of behaviors. The genomes are mutated and interbred creating new generations of robots according to a Darwinian scheme in which the fittest individuals are most likely to produce offspring. Fitness is measured in terms of how good a robot's behavior is according to some evaluation criteria; this is usually automatically measured but may, in the manner of eighteenth-century pig breeders, be based on the experimenters' judgment.

Motivation and Background

Turing's (1950) paper, *Computing Machinery and Intelligence*, is widely regarded as one of the seminal works in artificial intelligence. It is best known for what came to be called the Turing test – a proposal for deciding whether or not a machine is intelligent. However, tucked away toward the end of Turing's wide-ranging discussion of issues arising from the test is a far more interesting proposal. He suggests that worthwhile intelligent machines should be adaptive and should learn and develop but concedes that designing, building, and programming such machines by hand is probably completely infeasible. He goes on to sketch an alternative way of creating machines based on an artificial analog of biological evolution. Each machine would have hereditary material encoding its structure, mutated copies of which would form offspring machines. A selection mechanism would be used to favor better adapted machines – in this case, those that learned to behave most intelligently. Turing proposed that the selection mechanism should largely consist of the experimenter's judgment.

It was not until more than 40 years after their publication that Turing's long forgotten suggestions became reality. Building on the development of principled evolutionary search algorithm by, among others, Holland (1975), researchers at CNR, Rome, Case Western University, the University of Sussex, EPFL, and elsewhere independently demonstrated methodologies and practical techniques to evolve, rather than design, the control systems for primitive autonomous intelligent machines (Beer and Gallagher 1992; Cliff et al. 1993; de Garis 1990; Floreano and Mondada 1994; Husbands and Harvey 1992; Parisi and Nolfi 1993). Thus, the field of *Evolutionary Robotics* was born in the early 1990s. Initial motivations were similar to Turing's: the hand design of intelligent adaptive machines intended for operation in natural environments is extremely difficult, would it be possible to wholly or partly automate the process?

Today, the field of evolutionary robotics has expanded in scope to take in a wide range of applications, including promising new work on autonomous flying machines (Floreano et al. 2008; Vargas et al. 2014; Shim and Husbands 2007), as well as research aimed at exploring specific scientific issues – for instance, principles from neuroscience or questions in cognitive science (Harvey et al. 2005; Philippides et al. 2005; Floreano et al. 2008; Husbands et al. 2014). Such work is able to exploit the fact that evolutionary robotics operates with fewer assumptions about neural architectures and behavior-generating mechanisms than other methods; this means that whole general classes of designs and processes can be explored.

Structure of the Learning System

The key elements of the evolutionary robotics approach are the following:

- An artificial genetic encoding specifying the robot control systems, body plan, sensor properties, etc., along with a mapping to the target system
- A method for measuring the fitness of the robot behaviors generated from these genotypes
- A way of applying selection and a set of "genetic" operators to produce the next generation from the current

The structure of the overall evolutionary process is captured in Fig. 1. The general scheme is like that of any application of an evolutionary search algorithm. However, many details of specific parts of the process, particularly the evaluation step, are peculiar to evolutionary robotics.

The more general parts of the evolutionary process (selection, breeding, genetic operators such as mutation and crossover, replacement, and population structure) are also found in most other applications of evolutionary computing, and, just as in those other applications, there are many well-documented ways of implementing each (De Jong 2006; Eiben and Smith 2003). Hence, this section focuses on genetic encoding and evaluation as a route to more evolutionary robotics-specific issues. For a much fuller treatment of the subject, see Vargas et al. (2014), Doncieux et al. (2011), Floreano et al. (2008), and Nolfi and Floreano (2000).

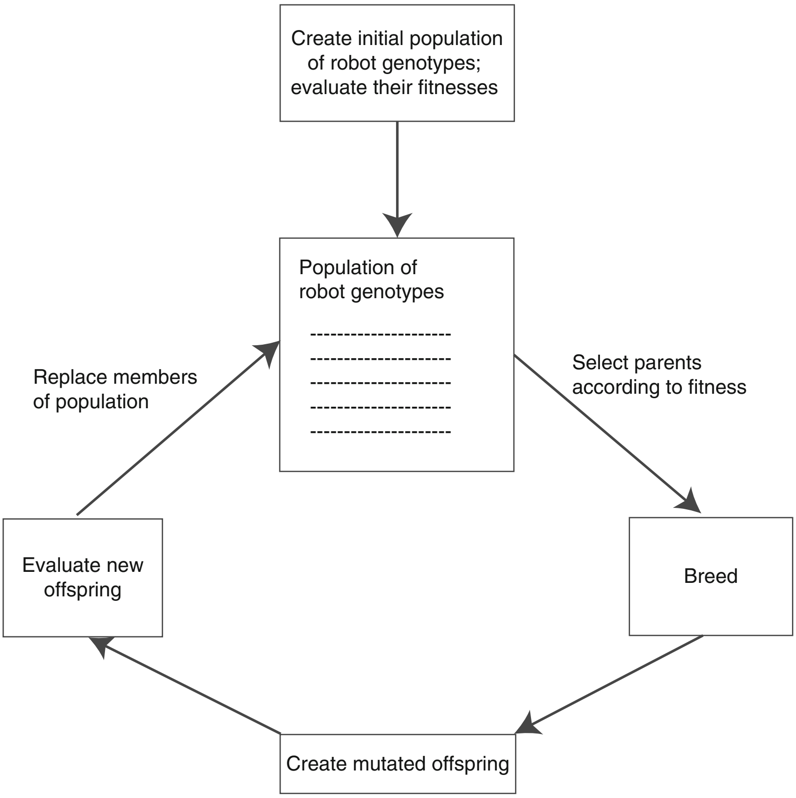
Genetic Encoding

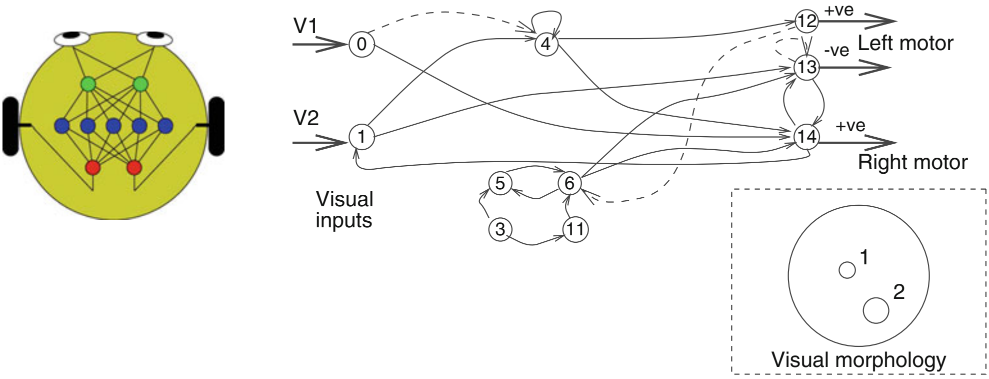
While, as already mentioned, many aspects of the robot design can potentially be under genetic control, *at least* the control system always is. By far the most popular form of controller is some sort of neural network. These range from straightforward feedforward networks of simple elements (Floreano and Mondada 1994) to relatively complex, dynamic, and plastic recurrent networks (Beer and Gallagher 1992; Floreano and Urzelai 2000; Philippides et al. 2005), as illustrated in Fig. 2. In the simplest case, a fixed architecture network is used to control a robot whose sensors feed into the network which in turn feeds out to the robot motors. In this scenario, the parameters of the network (connection weights and relevant properties of the units such as thresholds or biases) are coded as a fixed length string of numerical values.

A more complex case, which has been explored since the very early days of evolutionary robotics (Cliff et al. 1993), involves the evolution of the network architecture as well as the properties of the connections and units. Typically, the size of the network (number of units and connections) and its architecture (wiring diagram) are unconstrained and free to evolve. This involves more complex encodings which can grow and shrink, as units and connections are added or lost, while allowing a coherent decoding of connections between units. These range from relatively simple strings employing blocks of symbols that encode a unit's properties and connections relative to other units (Cliff et al.) to more indirect schemes that make use of developmental, growth processes in some geometric or topological space (Philippides et al. 2005; Stanley et al. 2009) or employ genetic programming-like tree representations in which whole subbranches can be added, deleted, or swapped over (Gruau 1995).

The most general case involves the encoding of control network and body and sensor properties. Various kinds of developmental schemes have been used to encode the construction of body morphologies from basic building blocks, both in simulation and in the real world. The position and properties of sensors can also be

Evolutionary Robotics, Fig. 1 General scheme employed in evolutionary robotics


Create initial population of robot genotypes; evaluate their fitnesses
Population of robot genotypes
Select parents according to fitness
Breed
Create mutated offspring
Evaluate new offspring
Replace members of population



Evolutionary Robotics, Fig. 2 Evolved neurocontrollers. On the *left* a simple fixed architecture feedforward network is illustrated. The connection weights, and sometimes the neuron properties, are put under evolutionary control. On the *right* a more complex architecture is illustrated. In this case, the whole architecture, including the number of neurons and connections, is under evolutionary control, along with connection and neuron properties and the morphology of a visual sensor that feeds into the network

put under evolutionary control. Sometimes one complex encoding scheme is used for all aspects of the robot under evolutionary control, and sometimes the different aspects are put on separate genotypes.

Fitness Evaluation

The fitness of members of the population is measured, via an evaluation mechanism, in terms of the robot behaviors produced by the control system or control system plus robot morphology that it encodes. Fitness evaluation, therefore, consists of translating the genome in question into a robot instantiation and then measuring the aspects of the resulting behavior. In the earliest work aimed at using evolutionary techniques to develop neurocontrollers for particular physical robots, members of a population were downloaded in turn onto the robot and their behavior was monitored and measured either automatically by clever experimental setups (Floreano and Mondada 1994; Harvey et al. 1994) or manually by an observer (Gruau and Quatramaran 1997). The machinery of the evolutionary search algorithm was managed on a host computer, while the fitness evaluations were undertaken on the target robot.

One drawback of evaluating fitness on the robot is that this cannot be done any quicker than in real time, making the whole evolutionary process rather slow. However, in the early work in the field, this approach was taken because it was felt that it was unlikely that simulations could be made accurate enough to allow proper transfer of evolved behavior onto the real robot. However, a careful study of accurate physics-based simulations of a Khepera robot, with various degrees of noise added, proved this assumption false (Jakobi et al. 1995). This led to the development of Jakobi's minimal simulation methodology (Jakobi 1998a), whereby computationally very efficient simulations are built by modeling only those aspects of the robot–environment interaction deemed important to the desired behavior and masking everything else with carefully structured noise (so that evolution could not come to rely on any of those features). These ultrafast, ultralean simulations have successfully been used with many different forms of robot and sensing, with very accurate transfer of behavior from simulation to reality. An alternative approach uses plastic controllers that further adapt through self-organization to help smooth out the differences between an inaccurate simulation and the real world (Urzelai and Floreano 2001). Instead of evolving connection weights, in this approach "learning rules" for adapting connection strengths are evolved – this results in controllers that continually adapt to changes in their environment. For details of further approaches, see Floreano et al. (2008). Much evolutionary robotics work now makes use of simulations; without them it would be impossible to do the most ambitious work on the concurrent evolution of controllers and body morphology (Lipson and Pollack 2000) (to be briefly described later). However, although simulation packages and techniques have developed rapidly in the past few years, there will still inevitably be discrepancies between simulation and reality, and the lessons and insights of the work outlined above should not be forgotten.

An interesting distinction can be made between implicit and explicit fitness functions in evolutionary robotics (Nolfi and Floreano 2000). In this context, an explicit fitness function rewards specific behavioral elements – such as traveling in a straight line – and hence shapes the overall behavior from a set of specific behavioral primitives. Implicit fitness functions operate at a more indirect, abstract level – fitness points are given for completing some task but they are not tied to specific behavioral elements. Implicit fitness functions might involve components such as maintaining energy levels or covering as much ground as possible, components that can be achieved in many different ways. In practice, it is quite possible to define a fitness function that has both explicit and implicit elements. Often fitness entails multiple and potentially conflicting elements, so methods from multi-objective optimisation have been introduced by some researchers, which can also encourage diversity in robot behavior (Mouret and Doncieux 2012).

Advantages

Potential advantages of this methodology include:

- The ability to explore potentially unconstrained designs that have large numbers of free variables. A *class* of robot systems (to be searched) is defined rather than specific, fully defined robot designs. This means fewer assumptions and constraints are necessary in specifying a viable solution.

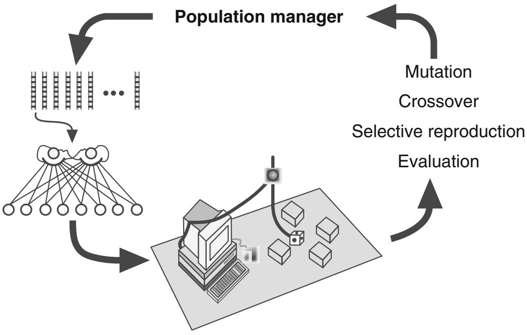
- The ability to use the methodology to fine-tune the parameters of an already successful design.
- The ability, through the careful design of fitness criteria and selection techniques, to take into account multiple, and potentially conflicting, design criteria and constraints (e.g., efficiency, cost, weight, power consumption, etc.).
- The possibility of developing highly unconventional and minimal designs.
- The ability to explicitly take into account robustness and reliability as major driving force behind the fitness measure, factors that are particularly important for certain applications.

Applications

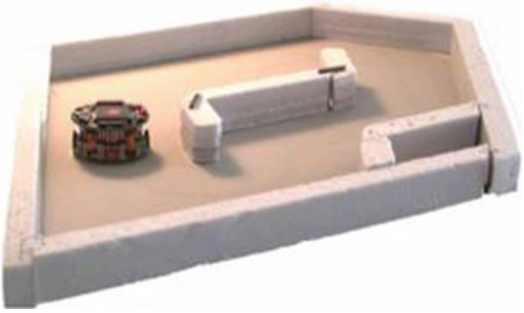
For a detailed survey of applications of evolutionary robotics, see Floreano et al. (2008) and Vargas et al. (2014); this section gives a brief overview of some areas covered by the methodology to give a better idea of the techniques involved and to indicate the scope of the field.

Prominent early centers for research in this area were EPFL and Sussex University, both of which are still very active in the field. Much of the early EPFL work used the miniature Khepera robot (Mondada et al. 1993), which became a popular tool in many areas of robotics research. In its simplest form, it is a two-wheeled cylindrical robot with a ring of IR sensors around its body. The first successful evolutionary robotics experiments at EPFL employed the setup illustrated in Figs. 3 and 4. A population of bit strings encoded the connection weights and node thresholds for a simple fixed architecture feedforward neural network. Each member of the population was decoded into a particular instantiation of a neural network controller which was then downloaded onto the robot (Floreano and Mondada 1994). This controlled the robot for a fixed period of time as it moved around the environment shown in Fig. 4.

The following simple fitness function was used to evolve obstacle avoidance behaviors:



Evolutionary Robotics, Fig. 3 Setup for early EPFL evolutionary robotics experiments with the Khepera robot (see text for details). Used with permission

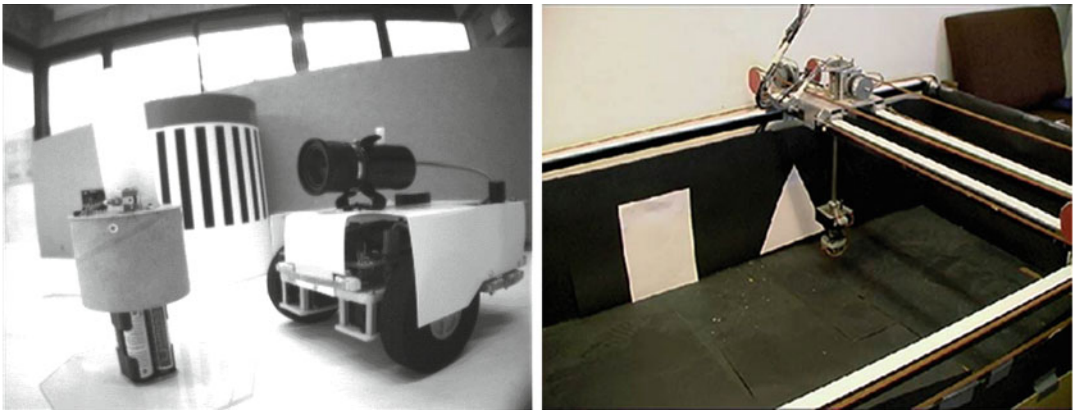


Evolutionary Robotics, Fig. 4 The simple environment used for evolving obstacle avoidance behaviors with a Khepera robot. Used with permission

$$F = V + (1 - \sqrt{DV}) + (1 - I)$$

where V is the average rotation speed of opposing wheels, DV is the difference between signed speed values of opposing wheels, and I is the activation value of the IR sensor with the highest input (readings are high if an obstacle is close to a sensor). Maximizing this function ensures high speed, a tendency to move in straight lines, and avoidance of walls and obstacles in the environment. After about 36 h of real-world evolution using this setup, controllers were evolved that successfully generated efficient motion around the course, avoiding collisions with the walls.

At the same time as this work was going on at EPFL, a series of pioneering experiments on evolving visually guided behaviors were being performed at Sussex University (Cliff et al. 1993; Harvey et al. 1994) in which discrete-



Evolutionary Robotics, Fig. 5 An early version of the Sussex gantry robot (*right*) was a "hardware simulation" of a robot such as that shown on the *left*. It allowed real-world evolution of visually guided behaviors in an easily controllable experimental setup (see text for further details)

time dynamical recurrent neural networks and visual sampling morphologies were concurrently evolved to allow a gantry robot (as well as other more standard mobile robots) to perform various visually guided tasks. An early instantiation of the Sussex gantry robot is shown in Fig. 5.

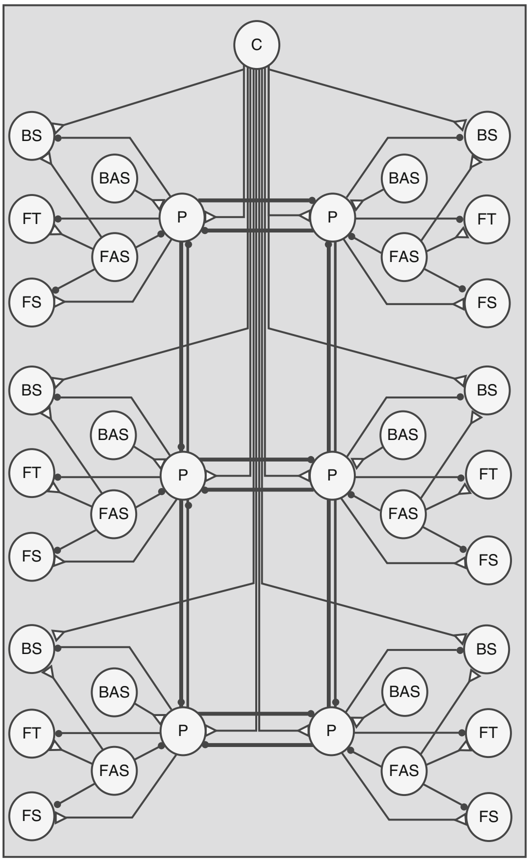
A CCD camera points down toward a mirror angled at 45°. The mirror can rotate around an axis perpendicular to the camera's image plane. The camera is suspended from the gantry allowing motion in the X, Y, and Z dimensions. This effectively provides an equivalent to a wheeled robot with a forward facing camera when only the X and Y dimensions of translation are used (see Fig. 5).

The apparatus was initially used in a manner similar to the real-world EPFL evolutionary robots setup illustrated in Fig. 3. A population of strings encoding robot controllers and visual sensing morphologies are stored on a computer to be downloaded one at a time onto the robot. The exact position and orientation of the camera head can be accurately tracked and used in the fitness evaluations. A number of visually guided navigation behaviors were successfully achieved, including navigating around obstacles and discriminating between different objects. In the experiment illustrated in Fig. 5, starting from a random position and orientation, the robot has to move to the triangle rather than the rectangle. This has to be achieved irrespective of the relative positions of the shapes and under very noisy lighting conditions. The architecture and all parameters of recurrent neural network controllers were evolved in conjunction with visual sampling morphologies – only genetically specified patches from the camera image were used (by being fed to input neurons according to a genetic specification), the rest of the image is thrown away. This resulted in extremely minimal systems only using two or three pixels of visual information yet still able to very robustly perform the task under highly variable lighting conditions. Behaviors were evolved in an incremental way, with more complex capabilities being evolved from populations of robots that were successful at some simpler task (for details see Harvey et al. 1994 and Harvey et al. 1997). The highly minimal yet very robust systems developed highlighted the potential for evolutionary robotics techniques in areas such as space exploration where there is a great pressure to minimize resources while maintaining reliability (Hobbs et al. 1996).

Since this early work, many different behaviors have been successfully evolved on a wide range of robots (Floreano et al. 2008; Nolfi and Floreano 2000; Vargas et al. 2014; Doncieux et al. 2011). There is not enough room to give an adequate summary of the whole field, so a few interesting subareas are highlighted below.

Over the past 15 years or so, there has been a growing body of work on evolving controllers for various kinds of walking robots – a nontrivial sensorimotor coordination task. Early work in this area concentrated on evolving dynamical network controllers for simple simulated insects (often inspired by cockroach studies), which were required to walk in uncomplicated environments (e.g., de Garis 1990; Beer and Gallagher 1992). The promise of this work soon led to versions of this methodology being used on real robots. Probably, the first success in this direction was by Lewis et al. (1992) who evolved a neural controller for a simple hexapod robot, using coupled oscillators built from continuous-time, leaky-integrator, artificial neurons. The robot was able to execute an efficient tripod gait on flat surfaces. All evaluations were done on the actual robot with each leg connected to its own pair of coupled neurons, leg swing being driven by one neuron, and leg elevation by the other. These pairs of neurons were cross-connected, in a manner similar to that used in the neural architecture shown in Fig. 6, to allow coordination between the legs. This architecture for locomotion, introduced by Beer et al. (1989), was based on the studies of cockroaches and has been much used ever since. Gallagher et al. (1996) used a generalization of it to evolve controllers for generating locomotion in a hexapod robot. This machine was more complex than Lewis et al.'s, with a greater number of degrees of freedom per leg. In this work, each leg was controlled by a fully connected network of five continuous-time, leaky-integrator neurons, each receiving a weighted sensory input from that leg's angle sensor. The connection weights and neuron time constants and biases were under genetic control. This produced efficient tripod gaits for walking on flat surfaces. In order to produce a wider range of gaits operating at a number of speeds such that rougher terrain could be successfully negotiated, a slightly different distributed architecture, more inspired by stick insect studies, was found to be more effective (Beer et al. 1997).

Jakobi (1998b) successfully used his minimal simulation techniques to evolve controllers for an eight-legged robot. Evolution in simulation took less than 2 h on what would today be regarded as a very slow computer and then transferred successfully to the real robot. Jakobi evolved modular controllers based on Beer's continuous recurrent network architecture to control the robot as it engaged in walking about its environment, avoiding obstacles and seeking out goals. The robot could smoothly change gait, move backward and forward, and even turn on the spot. More recently, related approaches have been successfully used to evolve controllers for more mechanically sophisticated robots such as the Sony Aibo (Tllez et al. 2006). In the last few years, there has also been successful work on evolving coupled oscillator style neural



Evolutionary Robotics, Fig. 6 Schematic diagram of a distributed neural network for the control of locomotion as used by Beer et al. Excitatory connections are denoted by *open triangles*, and inhibitory connections are denoted by *filled circles*. *C*, command neuron; *P*, pacemaker neuron; *FT*, foot motor neuron; *FS* and *BS*, forward swing and backward swing motor neurons; *FAS* and *BAS*, forward and backward angle sensors. Reproduced with permission controllers for the highly unstable dynamic problem of biped walking. Reil and Husbands (2002) showed that accurate physics-based simulations using physics engine software could be used to develop controllers able to generate successful bipedal gaits. Reil and colleagues have now significantly developed this technology to exploit its commercial possibilities in the animation and games industries (see www.naturalmotion.com for further details). Vaughan has taken related work in another direction. He has successfully applied evolutionary robotics techniques to evolve a simulation of a 3D ten degree of freedom bipedal robot. This machine demonstrates many of the properties of human locomotion. By using passive dynamics and compliant tendons, it conserves energy while walking on a flat surface. Its speed and gait can be dynamically adjusted and it is capable of adapting to discrepancies in both its environment and its body's construction (Vaughan et al. 2004). In general, the evolutionary development of neural network walking controllers, with their intricate dynamics, produces a wider range of gaits and generates smoother, more adaptive locomotion than the more standard use of finite state machine-based systems employing parameterized rules governing the timing and coordination of individual leg movements.

Early single robot research was soon expanded to handle interactions between multiple robots. Floreano and Nolfi did pioneering work on the coevolution of predator–prey behaviors in physical robots (Floreano et al. 2007). The fitness of the prey robot was measured by how quickly it could catch the prey; the fitness of the prey was determined by how long it could escape the predator. Two Khepera robots were used in this experiment, each had the standard set of proximity sensors but the predator also has a vision system, and the prey was able to move twice as fast as the predator. A series of interesting chasing and evasion strategies emerged. Later Quinn et al. (2003) demonstrated the evolution of coordinated cooperative behavior in a group of robots. A group of robots equipped only with IR proximity sensors were required to move as far as possible as a coordinated group starting from a random configuration. The task was solved by the robots adopting and then maintaining a specific formation. Analysis of the best evolved solution showed that it involved the robots adopting different roles, with the identical robots collectively "deciding" which robot would perform each role. Given the minimal sensing constraints, the evolved system would have proved extremely difficult to have designed by hand. For discussion of other multiple robot behaviors, see Floreano et al. (2008) and Vargas et al. (2014).

In the work described so far, control systems have been evolved for preexisting robots: the brain is constrained to fit a particular body and set of sensors. Of course in nature, the nervous system evolved simultaneously with the rest of the organism. As a result, the nervous system is highly integrated with the sensory apparatus and the rest of the body: the whole operates in a harmonious and balanced way – there are no distinct boundaries between the control system, the sensors, and the body.

Karl Sims started to explore the concurrent evolution of the brain and the body in his highly imaginative work involving simulated 3D "creatures" (Sims 1994). In this work, the creatures coevolved under a competitive scenario in which they were required to try and gain control of a resource (a cube) placed in the center of an arena. Both the morphology of the creatures and the neural system controlling their actuators were under evolutionary control.

Lipson and Pollack (2000), working at Brandeis University, pushed the idea of fully evolvable robot hardware about as far as was reasonably technologically feasible at the time. In an important piece of research, directly inspired by Sims' earlier simulation work, autonomous "creatures" were evolved in simulation out of basic building blocks (neurons, plastic bars, and actuators). The bars could connect together to form arbitrary truss structures with the possibility of both rigid and articulated substructures. Neurons could be connected to each other and to the bars whose length they would then control via a linear actuator. Machines defined in this way were required to move as far as possible in a limited time. The fittest individuals were then fabricated robotically

Evolutionary Robotics, Fig. 7 A fully automatically evolved robot developed on the Golem project (see text for details). Used with permission

using rapid manufacturing technology (plastic extrusion 3D printing) to produce results such as that shown in Fig. 7. They thus achieved autonomy of design and construction using evolution in a "limited universe" physical simulation coupled to automatic fabrication. The highly unconventional designs thus realized performed as well in reality as in simulation. The success of this work points the way to new possibilities in developing energy-efficient fault-tolerant machines.

Pfeifer and colleagues at Zurich University have explored issues central to the key motivation for fully evolvable robot hardware: the balanced interplay between body morphology, neural processing, and environment in the generation of adaptive behavior, and have developed a set of design principles for intelligent systems in which these issues take center stage (Pfeifer and Bongard 2007). Examples of interesting current work in this direction includes (Bongard 2011; Johnson et al. 2014).

Future Directions

Major ongoing challenges – methodological, theoretical, and technological – include finding the best way to incorporate development and lifetime plasticity within the evolutionary framework (this involves trends coming from the emerging field of epigenetic robotics), understanding better what the most useful building blocks are for evolved neurocontrollers, and finding efficient ways to scale work on concurrently evolving bodies and brains, especially in an open-ended way in the real world. For some grand challenges in the field, see (Eiben 2014).

There are very interesting developments in the evolution of group behaviors and the emergence of communication (Di Paolo 1998; Floreano et al. 2007; Quinn 2001; Vargas et al. 2014), the use of evolutionary robotics as a tool to illuminate problems in cognitive science (Beer 2003; Harvey et al. 2005) and neuroscience (Di Paolo 2003; Philippides et al. 2005; Seth 2005; Husbands et al. 2014), in developing flying behaviors (Floreano et al. 2007; Shim and Husbands 2007; Vargas et al. 2014), and in robots that have some form of self-model (Bongard et al. 2006), to name but a few.

Cross-References

▶ Neuroevolution

Recommended Reading

Beer RD (2003) The dynamics of active categorical perception in an evolved model agent (with commentary and response). Adapt Behav 11(4): 209–243

Beer RD, Chiel HJ, Sterling LS (1989) Heterogeneous neural networks for adaptive behavior in dynamic environments. In: Touretzky D (ed) Neural information processing systems, vol 1. Morgan Kauffman, San Francisco, pp 577–585

Beer RD, Gallagher JC (1992) Evolving dynamical neural networks for adaptive behaviour. Adapt Behav 1:94–110

Beer RD, Quinn RD, Chiel HJ, Ritzmann RE (1997) Biologically-inspired approaches to robotics. Commun ACM 40(3):30–38

Bongard J (2011) Morphological change in machines accelerates the evolution of robust behavior. Proc Natl Acad Sci 108(4):1234–1239

Bongard J, Zykov V, Lipson H (2006) Resilient machines through continuous self-modeling. Science 314:1118–1121

Cliff D, Harvey I, Husbands P (1993) Explorations in evolutionary robotics. Adapt Behav 2:73–110

de Garis H (1990) Genetic programming: evolution of time dependent neural network modules which teach a pair of stick legs to walk. In: Proceedings of the 9th European conference on artificial intelligence, Stockholm, pp 204–206

De Jong KA (2006) Evolutionary computation: a unified approach. MIT Press, Cambridge

Di Paolo E (1998) An investigation into the evolution of communication. Adapt Behav 6(2):285–324

Di Paolo EA (2003) Evolving spike-timing dependent plasticity for single-trial learning in robots. Philos Trans R Soc A 361:2299–2319

Doncieux S, Bredeche N, Mouret J-B (eds) (2011) New Horizons in evolutionary robotics: extended contributions from the 2009 EvoDeRob workshop. Studies in computational intelligence, vol 341. Springer, Berlin

Eiben AE (2014) Grand challenges for evolutionary robotics. Front Robot AI 1(4). doi:10.3389/frobt.2014.00004

Eiben AE, Smith JE (2003) Introduction to evolutionary computing. Springer, Berlin

Floreano D, Hauert S, Leven S, Zufferey JC (2007) Evolutionary swarms of flying robots. In: Floreano D (ed) Proceedings of the international symposium on flying insects and robots. EPFL, Monte Verita, pp 35–36

Floreano D, Husbands P, Nolfi S (2008) Evolutionary robotics. In: Siciliano B, Khatib O (eds) Springer handbook of robotics (Chap. 61). Springer, Berlin, pp 1423–1451

Floreano D, Mitri S, Magnenat S, Keller L (2007) Evolutionary conditions for the emergence of communication in robots. Curr Biol 17:514–519

Floreano D, Mondada F (1994) Automatic creation of an autonomous agent: genetic evolution of a neural-network driven robot. In: Cliff D, Husbands P, Meyer J, Wilson SW (eds) From animals to animats III: proceedings of the third international conference on simulation of adaptive behavior. MIT Press-Bradford Books, Cambridge, pp 402–410

Floreano D, Nolfi S (1997) Adaptive behavior in competing co-evolving species. In: Husbands P, Harvey I (eds) Proceedings of the 4th European conference on artificial life. MIT Press, Cambridge, pp 378–387

Floreano D, Urzelai J (2000) Evolutionary robots with on-line self-organization and behavioral fitness. Neural Netw 13(4–5):431–443

Gallagher J, Beer R, Espenschiel M, Quinn R (1996) Application of evolved locomotion controllers to a hexapod robot. Robot Auton Syst 19(1):95–103

Gruau F (1995) Automatic definition of modular neural networks. Adapt Behav 3(2):151–183

Gruau F, Quatramaran K (1997) Cellular encoding for interactive evolutionary robotics. In: Husbands P, Harvey I (eds) Proceedings of the 4th European conference on artificial life. The MIT Press/Bradford Books, Cambridge

Harvey I, Di Paolo E, Wood R, Quinn M, Tuci E (2005) Evolutionary robotics: a new scientific tool for studying cognition. Artif Life 11(1–2):79–98

Harvey I, Husbands P, Cliff DT (1994) Seeing the light: artificial evolution, real vision. In: Cliff DT, Husbands P, Meyer JA, Wilson S (eds) From animals to animats 3: proceedings of the third international conference on simulation of adaptive behaviour, SAB94. MIT Press, Cambridge, pp 392–401

Harvey I, Husbands P, Cliff D, Thompson A, Jakobi N (1997) Evolutionary robotics: the Sussex approach. Robot Auton Syst 20:205–224

Hobbs J, Husbands P, Harvey I (1996) Achieving improved mission robustness. In: 4th European Space Agency workshop on advanced space technologies for robot applications – ASTRA'96, Noordwijk, The Netherlands ESTEC

Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor

Husbands P, Harvey I (1992) Evolution versus design: controlling autonomous mobile robots. In: Proceedings of 3rd annual conference on artificial intelligence, simulation and planning in high autonomy systems. Computer Society Press, Los Alimitos, pp 139–146

Husbands P, Moioli R, Shim Y, Philippides A, Vargas P, O'Shea M (2014) Evolutionary robotics and neuroscience. In: Vargas P, Di Paolo E, Harvey I, Husbands P (eds.) The horizons of evolutionary robotics. MIT Press, Cambridge, pp 17–63

Jakobi N (1998a) Evolutionary robotics and the radical envelope of noise hypothesis. Adapt Behav 6:325–368

Jakobi N (1998b) Running across the reality gap: octopod locomotion evolved in a minimal simulation. In: Husbands P, Meyer JA (eds) Evolutionary robotics: first European workshop, EvoRobot98. Springer, Berlin, pp 39–58

Jakobi N, Husbands P, Harvey I (1995) Noise and the reality gap: the use of simulations in evolutionary robotics. In: Moran F et al (eds) Proceedings of 3rd European conference on artificial life. Springer, Berlin, pp 704–720

Johnson C, Philippides A, Husbands P (2014) Active shape discrimination with physical reservoir computers. In: Proceedings of Alife 14. MIT Press, Cambridge, pp 178–185

Lewis MA, Fagg AH, Solidum A (1992) Genetic programming approach to the construction of a neural network for a walking robot. In: Proceedings of IEEE international conference on robotics and automation. IEEE Press, Washington, pp 2618–2623

Lipson H, Pollack J (2000) Automatic design and manufacture of robotic lifeforms. Nature 406:974–978

Mondada F, Franzi E, Ienne P (1993) Mobile robot miniaturization: a tool for investigation in control algorithms. In: Yoshikawa T, Miyazaki F (eds) Proceedings of the third international symposium on experimental robotics. Springer, Berlin, pp 501–513

Mouret J-B, Doncieux S (2012) Encouraging behavioral diversity in evolutionary robotics: an empirical study. Evol Comput 20(1):91–133

Nolfi S, Floreano D (2000) Evolutionary robotics: the biology. In: Intelligence, and technology of self-organizing machines. MIT Press/Bradford Books, Cambridge

Parisi D, Nolfi S (1993) Neural network learning in an ecological and evolutionary context. In: Roberto V (ed) Intelligent perceptual systems. Springer, Berlin, pp 20–40

Pfeifer R, Bongard J (2007) How the body shapes the way we think: a new view of intelligence. MIT Press, Cambridge

Philippides A, Husbands P, Smith T, O'Shea M (2005) Flexible couplings: diffusing neuromodulators and adaptive robotics. Artif Life 11(1&2):139–160

Quinn M (2001) Evolving communication without dedicated communication channels. In: Kelemen J, Sosik P (eds) Proceedings of the 6th European conference on artificial life, ECAL'01. Springer, Berlin, pp 357–366

Quinn M, Smith L, Mayley G, Husbands P (2003) Evolving controllers for a homogeneous system of physical robots: structured cooperation with minimal sensors. Philos Trans R Soc Lond Ser A: Math Phys Eng Sci 361:2321–2344

Reil T, Husbands P (2002) Evolution of central pattern generators for bipedal walking in real-time physics environments. IEEE Trans Evol Comput 6(2): 10–21

Seth AK (2005) Causal connectivity analysis of evolved neural networks during behavior. Netw Comput Neural Syst 16(1):35–54

Shim YS, Husbands P (2007) Feathered flyer: integrating morphological computation and sensory reflexes into a physically simulated flapping-wing robot for robust flight Manoeuvre. In: Proceedings of ECAL. LNCS, vol 4648. Springer, Berlin, pp 756–765

Sims K (1994) Evolving 3D morphology and behavior by competition. In: Brooks R, Maes P (eds) Proceedings of artificial life IV. MIT Press, Cambridge, pp 28–39

Stanley K, D'Ambrosio D, Gauci J (2009) A hypercube-based encoding for evolving large-scale neural networks. Artif Life 15(2):185–212

Tllez R, Angulo C, Pardo D (2006) Evolving the walking behaviour of a 12 DOF quadruped using a distributed neural architecture. In: 2nd international workshop on biologically inspired approaches to advanced information technology (Bio-ADIT'2006). LNCS, vol 385. Springer, Berlin, pp 5–19

Turing AM (1950) Computing machinery and intelligence. Mind 59:433–460

Urzelai J, Floreano D (2001) Evolution of adaptive synapses: robots with fast adaptive behavior in new environments. Evol Comput 9:495–524

Vargas P, Di Paolo E, Harvey I, Husbands P (2014) The horizons of evolutionary robotics. MIT Press, Cambridge

Vaughan E, Di Paolo EA, Harvey I (2004) The evolution of control and adaptation in a 3D powered passive dynamic walker. In: Pollack J, Bedau M, Husbands P, Ikegami T, Watson R (eds) Proceedings of the ninth international conference on the simulation and synthesis of living systems artificial life IX. MIT Press, Cambridge, pp 139–145

Evolving Neural Networks

▸ Neuroevolution

Example

▸ Instance

Example Space

▸ Instance Space

Example-Based Programming

▸ Inductive Programming

Expectation Maximization Clustering

Xin Jin[1] and Jiawei Han[2]
[1]PayPal Inc., San Jose, CA, USA
[2]University of Illinios at Urbana-Champaign, Urbana, IL, USA

Abstract

The expectation maximization (EM) based clustering is a probabilistic method to partition data into clusters represented by model parameters. Extensions to the basic EM algorithm include but not limited to the stochastic EM algorithm (SEM), the simulated annealing EM algorithm (SAEM), and the Monte Carlo EM algorithm (MCEM).

Synonyms

Mixture model

Definition

The expectation maximization (EM) algorithm (Dempster et al. 1977; Fraley and Raftery 1998) finds maximum likelihood estimates of parameters in probabilistic models. EM is an iterative method which alternates between two steps, expectation (E) and maximization (M). For clustering, EM makes use of the finite Gaussian mixtures model and estimates a set of parameters iteratively until a desired convergence value is achieved. The mixture is defined as a set of K probability distributions and each distribution corresponds to one cluster. An instance is assigned with a membership probability for each cluster.

The EM algorithm for partitional clustering works as follows:

1. Guess initial parameters of the models: mean and standard deviation (if using normal distribution model).
2. Iteratively refine the parameters with E and M steps. In the E step: compute the membership possibility for each instance based on the initial parameter values. In the M step: recompute the parameters based on the new membership possibilities.
3. Assign each instance to the cluster with which it has highest membership possibility.

Refer to Celeux and Govaert (1995) for details about the E and M steps for multivariate normal mixture models parameterized via the eigenvalue decomposition.

The EM algorithm for clustering becomes time consuming to compute for models with very large numbers of components, because the number of conditional probabilities associated with each instance is the same as the number of components in the mixture.

Extensions

There are many extensions to the EM-based clustering algorithm. Celeux et al. (1996) compared three different stochastic versions of the EM algorithm: the stochastic EM algorithm (SEM), the simulated annealing EM algorithm (SAEM), and the Monte Carlo EM algorithm (MCEM). SEM was shown to be efficient for locating significant maxima of the likelihood function. The classification EM (CEM) algorithm (Celeux and Govaert 1992) incorporates a classification step between the E-step and the M-step using a maximum a posteriori (MAP) principle. The K-means algorithm becomes a particular version of the CEM algorithm corresponding to the uniform spherical Gaussian model. Yang et al. (2012) proposed an EM clustering algorithm for Gaussian mixture models, which is robust to initialization and different cluster sizes with a schema to automatically obtain an optimal number of clusters.

Softwares

The following softwares have implementations of the EM clustering algorithm:

- Weka. Open Source Data Mining Software in Java (Hall et al. 2009), from Machine Learning Group at the University of Waikato:
  http://www.cs.waikato.ac.nz/ml/weka/index.html
- LNKnet Software. Written in C. A public domain software from MIT Lincoln Laboratory:
  http://www.ll.mit.edu/mission/communications/cyber/softwaretools/lnknet/lnknet.html
- EMCluster (Chen et al. 2012). R package. It provides EM algorithms and several efficient initialization methods for clustering of finite mixture Gaussian distribution with unstructured dispersion in both unsupervised and semi-supervised learning.
  http://cran.r-project.org/web/packages/EMCluster/

Recommended Reading

Celeux G, Govaert G (1992) A classification em algorithm for clustering and two stochastic versions. Comput Stat Data Anal 14(3):315–332

Celeux G, Govaert G (1995) Gaussian parsimonious clustering models. Pattern Recognit 28(5):781–793
Celeux G, Chauveau D, Diebolt J (1996) Stochastic versions of the em algorithm: an experimental study in the mixture case. J Stat Comput Simul 55(4):287–314
Chen W-C, Maitra R, Melnykov V (2012) EMCluster: EM algorithm for model-based clustering of finite mixture Gaussian distribution. R Package, http://cran.r-project.org/package=EMCluster
Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J R Stat Soc Ser B (Methodol) 39(1):1–38
Fraley C, Raftery AE (1998) How many clusters? Which clustering method? Answers via model-based cluster analysis. Comput J 41(8):578–588
Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The weka data mining software: an update. ACM SIGKDD Explor Newsl 11(1):10–18
Yang M-S, Lai C-Y, Lin C-Y (2012) A robust em clustering algorithm for Gaussian mixture models. Pattern Recognit 45(11):3950–3961

Expectation Propagation

Tom Heskes
Institute for Computing and Information
Sciences, Radboud University Nijmegen,
Nijmegen, The Netherlands

Synonyms

EP

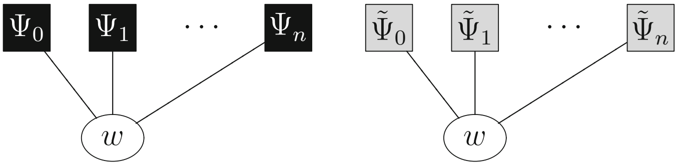
Definition

Expectation propagation is an algorithm for Bayesian machine learning. It tunes the parameters of a simpler approximate distribution (e.g., a Gaussian) to match the exact posterior distribution of the model parameters given the data. Expectation propagation operates by propagating messages, similar to the messages in (loopy) ▶ belief propagation. Whereas messages in belief propagation correspond to exact belief states, messages in expectation propagation correspond to approximations of the belief states in terms of expectations, such as means and variances. It is a deterministic method especially well suited to large databases and dynamic systems, where exact methods for Bayesian inference fail and ▶ Monte Carlo methods are far too slow.

Motivation and Background

One of the main problems for ▶ Bayesian methods is their computational expense: computation of the exact posterior given the observed data typically requires the solution of high-dimensional integrals that have no analytical expressions. Approximation algorithms are needed to approximate this posterior as accurately as possible. These techniques for approximate inference can be subdivided in two categories: deterministic approaches and stochastic sampling (Monte Carlo) methods. Having the important advantage that (under certain conditions) they give exact results in the limit of an infinite number of samples, ▶ Monte Carlo methods are the method of choice in Bayesian statistics. However, in particular, when dealing with large databases, the time needed for stochastic sampling to obtain a reasonably accurate approximation of the exact posterior can be prohibitive. This explains the need for faster, deterministic approaches, such as the Laplace approximation, ▶ variational approximations, and expectation propagation.

Expectation propagation was first described by Thomas Minka in his thesis Minka (2001). It can be viewed as a generalization and reformulation of the earlier ADATAP algorithm of Manfred Opper and Ole Winther (2001). Expectation propagation quickly became one of the most popular deterministic approaches for approximate Bayesian inference. Expectation propagation improves upon assumed density filtering, a classical method from stochastic control, by iteratively refining local approximations instead of computing them just once. Furthermore, it encompasses loopy belief propagation, a popular method for approximate inference in probabilistic graphical models, as a special case. Where loopy belief propagation is restricted to models of discrete variables only, expectation propagation applies to



Expectation Propagation, Fig. 1 (*left-hand side*) A so-called factor graph corresponding to the i.i.d. assumption in Bayesian machine learning. Each box corresponds to a factor or term. A *circle* corresponds to a variable. Factors are connected to the variables that they contain. Ψ_0 corresponds to the prior, and $\Psi_1 \ldots \Psi_n$ are the likelihood terms for the n data points. The *right-hand side* is a factor graph of the approximating distribution. The original terms have been replaced by term approximations

a much wider class of probabilistic ▸ graphical models with discrete and continuous variables and complex interactions between them.

Structure of Learning System

Bayesian Machine Learning

In the Bayesian framework for machine learning, you should enumerate all reasonable models of the data and assign a prior belief $P(w)$ to each of these models w. Then, upon observing the data D, you compute the likelihood $P(D|w)$ to evaluate how probable the data was under each of these models. The product of the prior and the likelihood gives you, up to a normalization constant, the posterior probability $P(w|D)$ over models given the data:

$$P(w|D) = \frac{P(D|w)P(w)}{P(D)} \, ,$$

where the normalization term $P(D)$ is called the probability of the data or "evidence." This posterior probability incorporates all you have learned from the data D regarding the models w under consideration. As indicated above, exact calculation of this posterior probability is often infeasible, because the normalization term requires the solution of intractable sums or integrals.

In its simplest setting, the data D consists of n observations, $x_1, \ldots, x_n$, which are assumed to be i.i.d. (independent and identically distributed). The posterior probability then factorizes into $n + 1$ terms, one for each observation and one for the prior. With definitions $\Psi_0(w) \equiv P(w)$ and $\Psi_i(w) \equiv P(x_i|w)$, we can rewrite

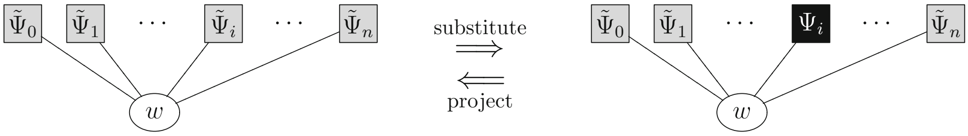
$$P(w|D) = \frac{P(w)\prod_{i=1}^{n} P(x_i|w)}{P(D)} \equiv \frac{\prod_{i=0}^{n} \Psi_i(w)}{P(D)} \, .$$

This factorization is visualized in the so-called factor graph in Fig. 1. We will use it as a running example in the following.

Assumed Density Filtering

Expectation propagation can be interpreted by an iterative refinement of assumed density filtering. In assumed density filtering, we add terms one by one and project in each step back to the "assumed density." For example, suppose that our prior probability $P(w) = \Psi_0(w)$ is a (known) Gaussian distribution over model parameters w, the terms corresponding to the data points are non-Gaussian, and we aim to find an appropriate Gaussian approximation $Q(w)$ to the exact (non-Gaussian) posterior $P(w|D)$. Our first approximation is the prior itself. Assumed-density filtering now proceeds by adding terms one at a time, where at each step we approximate the resulting distribution as closely as possible by a Gaussian. The pseudo-code is given in Algorithm 1, where $Q_{0:i}(w)$ denotes the approximation obtained after incorporating the prior and the first i observations.

If we use the Kullback-Leibler divergence as the distance measure from the non-Gaussian (but normalized) product of $Q_{0:i-1}(w)$ and $\Psi_i(w)$ and the Gaussian approximation, projection becomes "moment matching": the result of the projection



Expectation Propagation, Fig. 2 Visualization of expectation propagation when recomputing the term approximation for observation i

is the Gaussian that has the same mean and covariance matrix as the non-Gaussian product.

Expectation Propagation

When in assumed density filtering we add the term $\Psi_i(w)$, the Gaussian approximation changes from $Q_{0:i-1}(w)$ to $Q_{0:i}(w)$. We will call the quotient of the two the *term approximation* (here and in the following we ignore normalization constants):

$$\tilde{\Psi}_i(w) = \frac{Q_{0:i}(w)}{Q_{0:i-1}(w)} .$$

In our running example, term approximations are quotients between two different Gaussian densities and therefore have a Gaussian form themselves. Since the prior $\Psi_0(w)$ is a Gaussian density, $\tilde{\Psi}_0(w) = \Psi_0(w)$. The approximation $Q_{0:n}(w)$ is equal to the product of all term approximations and is visualized on the right-hand side of Fig. 1. In assumed density filtering, the resulting approximation depends on the ordering in which the terms have been added. For example, if the terms had been added in reverse order, their term approximations might have been (slightly) different.

Expectation propagation now generalizes assumed density filtering by iteratively refining these term approximations. When successful, the final approximation will be independent of the ordering. Pseudo-code of expectation propagation is given in Algorithm 2. In step 1 through 5, the term approximations are initialized; in step 6 through 12, these term approximations are iteratively refined until they no longer change. In step 8, we take out the previous term approximation from the current approximation. In step 9, we put back in the exact term and project back to a Gaussian, like we did in assumed density filtering. It is easy to check that the approximation $Q(w)$ after the first loop equals the approximation $Q_{0:n}(w)$ obtained with assumed density filtering. The recalculation of the term approximation corresponding to observation i is visualized in Fig. 2.

Computational Aspects

With expectation propagation, we have to do a little more bookkeeping than with assumed density filtering: we have to keep track of the term approximations. One loop of expectation propagation is about as expensive as running assumed density filtering. Typically, about five iterations are sufficient for convergence.

The crucial operation is in step 3 of Algorithm 1 and step 9 of Algorithm 2. Here we have to compute the moments of the (non-Gaussian) probability distribution on the right-hand side. In most cases, we do not have analytical expressions for these moments and have to compute them numerically, e.g., using Gaussian quadrature. We then obtain the moments (mean and covariance matrix) of the new approximation $Q(w)$. Divisions and multiplications correspond to a simple subtraction and addition of so-called canonical parameters. For the Gaussian these canonical parameters are the inverse of the covariance matrix (precision matrix) and the product of the precision matrix and the mean. The bottom line is that we go back and forth between distributions in terms of moments and in terms of canonical parameters. For a Gaussian, this requires computing the inverse of the covariance matrix, which is roughly on the order of d^3, where d is the dimension of w. A practical point of concern is that matrix inversion is numerically instable, in particular, for matrices that are close to singular, which can lead to serious roundoff errors.

Algorithm 1 Assumed density filtering

```
$Q_0(w) = \Psi_0(w)$
for $i = 1$ to $n$ do
    $Q_{0:i}(w) =$ Project_to_Gaussian($Q_{0:i-1}(w)\Psi_i(w)$)
end for
```

Algorithm 2 Expectation propagation

```
$\tilde{\Psi}_0(w) = \Psi_0(w)$
for $i = 1$ to $n$ do
    $\tilde{\Psi}_i(w) = 1$
end for
$Q(w) = \prod_{i=0}^{n} \tilde{\Psi}_i(w)$
while not converged do
    for $i = 1$ to $n$ do
        $Q_{-i}(w) = \frac{Q(w)}{\tilde{\Psi}_i(w)}$
        $Q(w) =$ Project_to_Gaussian($Q_{-i}(w)\Psi_i(w)$)
        $\tilde{\Psi}_i(w) = \frac{Q(w)}{Q_{-i}(w)}$
    end for
end while
```

Convergence Issues

Sadly enough, expectation propagation is not guaranteed to converge to a fixed point. If it does, this fixed point can be shown to correspond to an extremum of the so-called Bethe free energy, an approximation of the "evidence" $\log P(D)$, under particular consistency and normalization constraints (Minka 2001; Herbrich and Graepel 2006; Heskes and Zoeter 2002; Heskes et al. 2005). These constraints relate to the projection step in Algorithm 2: after convergence, the moments of $Q(w)$ should be equal to the moments of the distribution obtained by taking out a term approximation and putting back the corresponding exact term. This should hold for all i.i.d. observations $i = 1, \ldots, n$ in the factor graph of Fig. 1: so we conclude that, after convergence, the moments ("expectations") of all distributions constructed in this way should be the same. Expectation consistent approximations are based on the exact same idea and indeed turn out to be equivalent to expectation propagation (Heskes et al. 2005).

When expectation propagation does not converge, we can try "damping": instead of replacing the old term approximation by the new one, we replace it by a logconvex combination of the old and the new one. In many cases, damping with a step size 0.1 makes expectation propagation converge, at the expense of requiring more iterations. However, even damping with an infinitesimally small step size is not guaranteed to lead to convergence. In those cases, we can try to minimize the Bethe free energy more explicitly with a so-called double-loop algorithm (Heskes and Zoeter 2002): in the outer loop we compute a convex bound on the Bethe free energy, which we then minimize in the inner loop with an algorithm very similar to standard expectation propagation. Double-loop algorithms are an order of magnitude slower than standard expectation propagation. Recent approaches such as Seeger and Nickisch (2010) provide guaranteed convergence at a much faster rate, but only for specific models.

Generalizations

The running example above serves to illustrate the main idea, but is of course rather restrictive. Expectation propagation can be applied with any member of the exponential family as approximating distribution (Minka 2001; Seeger 2008). The crucial operations are the projection step and the transformation from moment to canonical form: if these can be performed efficiently and robustly, expectation propagation is into play.

In many interesting cases, the model to be learned (here represented as a single variable w) contains a lot of structure. This structure can be exploited by expectation propagation to make it more efficient. For example, when a term only contains a subset of the elements of w, so does its term approximation. Also, we might take as the approximating distribution a distribution that factorizes over the elements of w, instead of a "full" distribution coupling all elements. For a Gaussian, this would amount to a diagonal instead of a full covariance matrix. Such a factorization will lead to lower memory requirements and faster computation, perhaps at the expense of reduced accuracy. More advanced approximations include Tree-EP, where the approximating structure is a tree, and generalized expectation propagation, which generalizes expectation propagation to include higher-order interactions in the same way as generalized belief propagation generalizes loopy belief propagation (Welling et al. 2005). Systematic higher-order corrections on top of standard expectation propagation lead to improved approximate inference in Gaussian latent variable models (Cseke and Heskes 2011; Opper et al. 2013).

Power expectation propagation (Minka 2005) generalizes expectation propagation by considering a different distance measure in the projection step. Instead of taking the Kullback-Leibler divergence, we can take any so-called α-divergence. $\alpha = 1$ corresponds to the Kullback-Leibler divergence and $\alpha = -1$ to the Kullback-Leibler divergence with the two probabilities interchanged. In the latter case, we obtain a variational method called variational Bayes.

Programs and Data

Code for expectation propagation applied to Gaussian process classification can be found at http://www.gaussianprocess.org/gpml/code/matlab/doc/ or http://becs.aalto.fi/en/research/bayes/gpstuff/. Kevin Murphy's Bayes Net toolbox (https://code.google.com/p/bnt/) can provide a good starting point to write your own code for expectation propagation. Expectation propagation is one of the approximate inference methods implemented in Infer.NET, Microsoft's framework for running Bayesian inference in graphical models (http://research.microsoft.com/en-us/um/cambridge/projects/infernet/).

Applications

Expectation propagation has been applied for, among others, Gaussian process classification (Csato et al. 2002), inference in Bayesian networks and Markov random fields, text classification with Dirichlet models and processes (Minka and Lafferty 2002), logistic regression models for rating players (Herbrich and Graepel 2006), and inference and learning in hybrid and nonlinear dynamic Bayesian networks (Heskes and Zoeter 2002).

Future Directions

From an application point of view, expectation propagation will probably become one of the standard techniques for approximate Bayesian machine learning, much like the Laplace approximation and Monte Carlo methods. Future research may involve questions like

- When does expectation propagation converge? Can we design variants that are guaranteed to converge for any type of model?
- What "power" to use in power expectation propagation for what kind of purposes?
- Can we adapt expectation propagation to handle approximating distributions that are not part of the exponential family? Recent progress in this direction includes Barthelme and Chopin (2014).

Cross-References

▶ Gaussian Process

Recommended Reading

Barthelmé S, Chopin N (2014) Expectation propagation for likelihood-free inference. J Am Stat Assoc 109(505):315–333

Csató L (2002) Gaussian processes – iterative sparse approximations. Ph.D. thesis, Aston University

Cseke B, Heskes TT (2011) Approximate marginals in latent Gaussian models. J Mach Learn Res 12:417–454

Herbrich R, Graepel T (2006) TrueSkill: a Bayesian skill rating system. Technical report (MSR-TR-2006-80), Microsoft Research, Cambridge

Heskes T, Zoeter O (2002) Expectation propagation for approximate inference in dynamic Bayesian networks. In: Darwiche A, Friedman N (eds) Proceedings of the 18th conference on uncertainty in artificial intelligence, Alberta, pp 216–223

Heskes T, Opper M, Wiegerinck W, Winther O, Zoeter O (2005) Approximate inference with expectation constraints. J Stat Mech Theory Exp P11015
Minka T (2001) A family of algorithms for approximate Bayesian inference. Ph.D. thesis, MIT
Minka T (2005) Divergence measures and message passing. Technical report (MSR-TR-2005-173), Microsoft Research, Cambridge
Minka T, Lafferty J (2002) Expectation-propagation for the generative aspect model. In: Darwiche A, Friedman N (eds) Proceedings of the 18th conference on uncertainty in artificial intelligence, Alberta, pp 352–359
Opper M, Paquet U, Winther O (2013) Perturbative corrections for approximate inference in Gaussian latent variable models. J Mach Learn Res 14(1):2857–2898
Opper M, Winther O (2001) Tractable approximations for probabilistic models: the adaptive Thouless-Anderson-Palmer mean field approach. Phys Rev Lett 86:3695–3699
Seeger M (2008) Bayesian inference and optimal design for the sparse linear model. J Mach Learn Res 9: 759–813
Seeger M, Nickisch H (2010) Fast convergent algorithms for expectation propagation approximate Bayesian inference. arXiv preprint arXiv: 1012.3584
Welling M, Minka T, Teh Y (2005) Structured region graphs: morphing EP into GBP. In: Bacchus F, Jaakkola T (eds) Proceedings of the 21st conference on uncertainty in artificial intelligence (UAI), Edinburgh, pp 609

Experience Curve

► Learning Curves in Machine Learning

Experience-Based Reasoning

► Case-Based Reasoning

Explanation

In ► Minimum Message Length, an *explanation is* a code with two parts, where the first part is an *assertion* code and the second part is a *detail* code.

Explanation-Based Generalization for Planning

► Explanation-Based Learning for Planning

Explanation-Based Learning

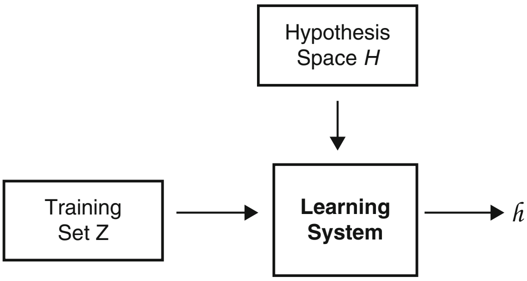
Gerald DeJong[1] and Shiau Hong Lim[2]
[1]University of Illinois at Urbana, Urbana, IL, USA
[2]University of Illinois, Champaign, IL, USA

Synonyms

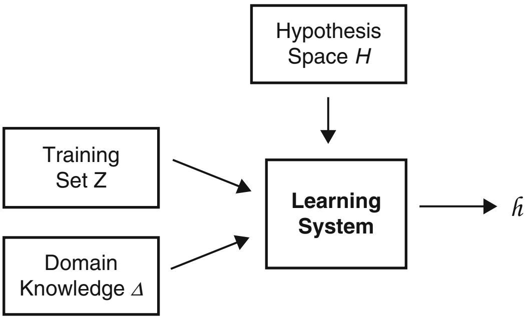
Analytical learning; Deductive learning; EBL; Utility problem

Definition

Explanation-based learning (EBL) is a principled method for exploiting available domain knowledge to improve ► supervised learning. Improvement can be in speed of learning, confidence of learning, accuracy of the learned concept, or a combination of these. In modern EBL the domain theory represents an expert's approximate knowledge of complex systematic world behavior. It may be imperfect and incomplete. Inference over the domain knowledge provides *analytic* evidence that compliments the empirical evidence of the training data. By contrast, in original EBL, the domain theory is required to be much stronger; inferred properties are guaranteed. Another important aspect of modern EBL is the interaction between domain knowledge and labeled training examples afforded by explanations. Interaction allows the nonlinear combination of evidence so that the resulting information about the target concept can be much greater than the sum of the information from each evidence source taken independently.



Explanation-Based Learning, Fig. 1 Conventional learner



Explanation-Based Learning, Fig. 2 EBL learner

Motivation and Background

A conventional machine learning system is illustrated in Fig. 1. A hypothesis $\hat{h}$ is selected from a space of candidates H using a training set of labeled examples Z as evidence. It is common to assume that the examples are drawn from some space of well-formed inputs X according to some fixed but unknown distribution $\mathcal{D}$. The quality of $\hat{h}$ is to be judged against different examples similarly selected and labeled. The correct label for an example is specified by some ideal *target concept*, c^*. This is typically some complex world process whose outcome is of interest. The target concept, c^*, will generally not be a member of space of acceptable candidates, H. Rather, the learner tries to find some $\hat{h}$ which is acceptably similar to c^* over $X_{\mathcal{D}}$ and can serve as a computationally tractable stand-in.

Of course, good performance of $\hat{h}$ on Z (its training performance) alone is insufficient. The learner must achieve some statistical guarantee of good performance on the underlying distribution (test performance). If H is too rich and diverse or if Z is too impoverished, a learner is likely to ▸ overfit the data; it may find a pattern in the training data that does not hold in the underlying distribution $X_{\mathcal{D}}$. Test performance will be poor despite good training performance.

An explanation-based learner employs its domain theory, Δ (Fig. 2), as an additional source of information. This domain theory must not be confused with ▸ learning bias, which is present in all learners. Determinations (Russell and Grosof 1987) provide an extreme illustration. These are logical expressions that make strong claims about the world but only after seeing a training example. EBL domain theories are used only to explain. An inferred expression is not guaranteed to hold but only provides analytic evidence.

An explanation for some $z \in Z$ is immediately and easily generalized: The structure of the explanation accounts for why zs assigned classification label should follow from its features. All other examples that meet these conditions are assigned the same classification by the generalized explanation for the same reasons.

Early approaches to EBL (e.g., DeJong and Mooney 1986; Mitchell et al. 1986; Mitchell 1997; Russell and Norvig 2003) were undone by two difficult problems: (1) unavoidable imperfections in the domain theory and (2) the utility problem. The former stems from assuming a conventional semantics for the domain theory. It results in a brittleness and an under-reliance on the training data. Modern EBL is largely a reaction to this difficulty. The utility problem is a consequence of an ill-defined hypothesis space and, as will be discussed later, can be avoided in a straightforward manner.

Structure of Learning System

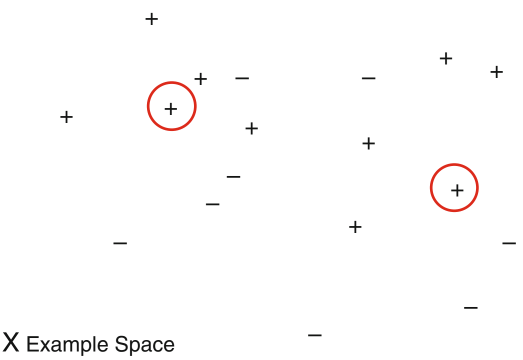
Explanations and Their Generalization

An *explanation* for a training example is any causal structure, derivable from Δ, which justifies why this training example might merit its teacher-assigned classification label. A *generalized explanation* is the structure of an explanation without the commitment to any particular example. The explanation and generalization processes are relatively straightforward and not significantly different from the original EBL algorithms.

The weakness of early EBL is in viewing the components of Δ as constraints. This leads to a view of explanations and their generalizations as *proofs*. Real-world brittleness due to the qualification problem (McCarthy 1980) follows inevitably. In modern EBL, Δ is seen as approximating the underlying world constraints (DeJong 2006; Kimmig et al. 2007). The domain theory is fundamentally a statistical device. Its analytic evidence and the empirical evidence of the training examples both provide a bridge to the real world.

The domain theory introduces new predicates and specifies their significant potential interactions. From a statistical point of view, these are named latent (hidden) features together with a kind of grammar for constructing alternative estimators for them. In short, the domain theory compactly specifies a large set of conceptual structures that an expert believes may be useful in making sense of the domain. If the expert is correct, then patterns of interest will become computationally much more accessible via analytic inference.

One flexible and useful form of a domain theory is sound inference over a set of first-order symbolic logic sentences. In such domain theories, the explanation mechanism can be identical to logical deduction although using a paraconsistent inference mechanism; inference must be well behaved despite inconsistencies in the theory. Generalized explanations are simply "theorems" of Δ that relate a classification label to the values of observable example features. But since the sentences of the theory only approximate world constraints, derivation alone, even via sound inference, is not sufficient evidence to believe a conclusion. Thus, a generalized explanation is only a conjecture. Additional training examples beyond those used to generate each explanation help to estimate the utility of these generalizations.



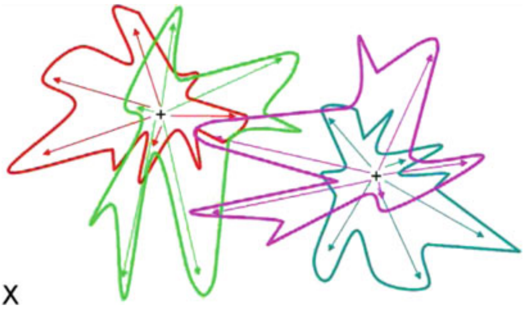
Explanation-Based Learning, Fig. 3 An example space with two designated positive training items

But analytic mechanisms need not be limited to symbolic logic-like inference. For example, one EBL approach is to distinguish handwritten Chinese characters (Lim et al. 2007) employing a Hough transform as a component of the domain theory. There, an explanation conjectures (hidden) glyph "strokes" to explain how the observed pixels of the training images may realize the image's character label.

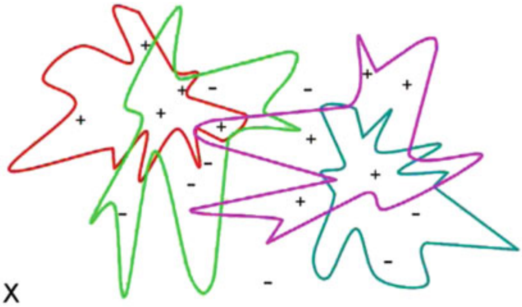
Whatever the form of the analytic inferential mechanism, multiple, quite incompatible explanations can be generated; the same training label can be explained using different input features and postulating different interactions. Such explanations will generalize to cover quite different subsets of X. Figure 3 shows a small training set with two positive examples highlighted. While the explanation process can be applied to all examples both positive and negative, these two will be used to illustrate. In this illustration, just two explanations are constructed for each of the highlighted training examples. Figure 4 shows the generalized extensions of these four explanations in the example space. The region enclosed by each contour is meant to denote the subset of X conjectured to merit the same classification as the explained example. Explanations make no claim about the labels for examples outside their extension.

Evaluation and Hypothesis Selection

Additional training examples that fall within the extension of a generalized explanation help to evaluate it empirically. This is shown in Fig. 5. The estimated utility of a generalized explanation



Explanation-Based Learning, Fig. 4 Four constructed explanations are sufficient to cover the positive examples



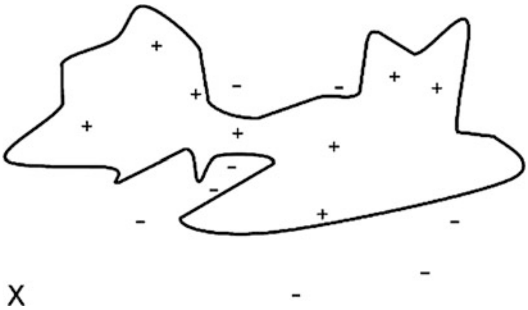
Explanation-Based Learning, Fig. 5 Explanations are evaluated with other training examples

reflects (1) the generalized explanation's empirical accuracy on these training examples, (2) the inferential effort required to derive the explanation (see DeJong 2006), and (3) the redundancies and interactions with other generalized explanations (higher utility is estimated if its correct predictions are less commonly shared by other generalized explanations).

The estimated utilities define an EBL classifier as a mixture of the generalized explanations each weighted by its estimated utility:

$$\hat{c}_{\mathrm{EBL}}(x) = \sum_{g \in \mathrm{GE}(Z,\Delta)} u_g \cdot g(x),$$

where $\mathrm{GE}(Z, \Delta)$ denotes the generalized explanations for Z from Δ and u_g is the estimated utility for g. This corresponds to a voting scheme where each generalized explanation that claims to apply to an example casts a vote in proportion to its estimated utility. The votes are normalized



Explanation-Based Learning, Fig. 6 An element from H that approximates the weighted explanations

over the utilities of voting generalized explanations. The mixture scheme is similar to that of sleeping experts (Freund et al. 1997). This EBL classifier approximates the target concept c^*. But unlike the approximation chosen by a conventional learner, $\hat{c}_{\mathrm{EBL}}$ reflects the information of Δ in addition to Z.

The final step is to select a hypothesis $\hat{h}$ from H. The EBL concept $\hat{c}_{\mathrm{EBL}}$ is used to guide this choice. Figure 6 illustrates the selection of a $\hat{h} \in H$, which is a good approximation to a utility-blended mixture of Fig. 5. This final step, selecting a hypothesis from H, is important but was omitted in original EBL. These systems employed generalized explanations directly. Unfortunately, such classifiers suffer from a difficulty known as the *utility problem* (Minton 1990). Note this is a slightly different use of the term *utility*, referring to the performance of an application system. This system can be harmed more than helped by concepts such as $\hat{c}_{\mathrm{EBL}}$, even if these concepts provide highly accurate classification. Essentially, the average cost of evaluating an EBL concept may outweigh the average benefit that it provides to the application system. It is now clear that this utility problem is simply the manifestation of a poorly structured hypothesis space. Note that, in general, an EBL classifier itself will not be an element of the space of acceptable hypotheses H. Previous approaches to the utility problem (Minton 1990; Gratch and DeJong 1992; Greiner and Jurisica 1992; Etzioni 1993) identify and disallow offending EBL concepts. However, the root cause is addressed by employing the EBL concept as a guidance in selecting a $\hat{h} \in H$

rather than using $\hat{c}_{\mathrm{EBL}}$ directly. Without this last step, H is completely ignored. But H embodies all of the information in the learning problem about what makes an acceptable hypothesis. The "utility problem" is simply the manifestation of leaving out this important information.

Literature

The roots and motivation for EBL extend at least to the MACROPs of STRIPS (Fikes et al. 1972). The importance of explanations of training examples was first suggested in DeJong (1981). The standard references for the early EBL work are Mitchell et al. (1986) and DeJong and Mooney (1986). When covering EBL, current textbooks give somewhat refined versions of this early approach (Mitchell 1997; Russell and Norvig 2003). Important related ideas include determinations (Russell and Grosof 1987), chunking (Laird et al. 1986), and knowledge compilation (Anderson 1986). EBL's ability to employ first-order theories make it an attractive compliment to learning Horn theories with ▶ Inductive Logic Programming (Hirsh 1987; Bruynooghe et al. 1989; Pazzani and Kibler 1992; Zelle and Mooney 1993). The problem of imperfect domain theories was recognized early, and there have been many approaches (Flann and Dietterich 1989; Genest et al. 1990; Towell et al. 1991; Cohen 1992; Thrun and Mitchell 1993; Ourston and Mooney 1994). But with modern statistical learning ascending to the dominant paradigm of the field, interest in analytic approaches waned. The current resurgence of interest is largely driven by placing EBL in a modern statistically sophisticated framework that nonetheless is still able to exploit a first-order expressiveness (DeJong 2006; Kimmig et al. 2007; Lim et al. 2007; Sun and DeJong 2005).

Cross-References

▶ Deductive Learning
▶ Explanation-Based Learning for Planning
▶ Speedup Learning

Recommended Reading

Anderson J (1986) Knowledge compilation: the general learning mechanism. In: Michalski R, Carbonell J, Mitchell T (eds) Machine learning II. Morgan Kaufmann, San Mateo, pp 289–310

Bruynooghe M, De Raedt L, De Schreye D (1989) Explanation based program transformation. In: IJCAI'89: proceedings of the eleventh international joint conference on artificial intelligence, Detroit, pp 407–412

Cohen WW (1992) Abductive explanation-based learning: a solution to the multiple inconsistent explanation problem. Mach Learn 8:167–219

DeJong G (1981) Generalizations based on explanations. In: IJCAI'81: proceedings of the seventh international joint conference on artificial intelligence, Vancouver, pp 67–69

DeJong G (2006) Toward robust real-world inference: a new perspective on explanation-based learning. In: ECML06: proceedings of the seventeenth European conference on machine learning, Berlin. Springer, Heidelberg, pp 102–113

DeJong G, Mooney R (1986) Explanation-based learning: an alternative view. Mach Learn 1(2):145–176

Etzioni O (1993) A structural theory of explanation-based learning. Artif Intell 60(1):93–139

Fikes R, Hart PE, Nilsson NJ (1972) Learning and executing generalized robot plans. Artif Intell 3(1–3):251–288

Flann NS, Dietterich TG (1989) A study of explanation-based methods for inductive learning. Mach Learn 4:187–226

Freund Y, Schapire RE, Singer Y, Warmuth MK (1997) Using and combining predictors that specialize. In: Twenty-ninth annual ACM symposium on the theory of computing, El Paso, pp 334–343

Genest J, Matwin S, Plante B (1990) Explanation-based learning with incomplete theories: a three-step approach. In: Proceedings of the seventh international conference on machine learning, Austin, pp 286–294

Gratch J, DeJong G (1992) Composer: a probabilistic solution to the utility problem in speed-up learning. In: AAAI, San Jose, pp 235–240

Greiner R, Jurisica I (1992) A statistical approach to solving the EBL utility problem. In: National conference on artificial intelligence, San Jose, pp 241–248

Hirsh H (1987) Explanation-based generalization in a logic-programming environment. In: IJCAI'87: proceedings of the tenth international joint conference on artificial intelligence, Milan, pp 221–227

Kimmig A, De Raedt L, Toivonen H (2007) Probabilistic explanation based learning. In: ECML'07: proceedings of the eighteenth European conference on machine learning, Warsaw, pp 176–187

Laird JE, Rosenbloom PS, Newell A (1986) Chunking in soar: the anatomy of a general learning mechanism. Mach Learn 1(1):11–46

Lim SH, Wang L-L, DeJong G (2007) Explanation-based feature construction. In: IJCAI'07: proceedings of the twentieth international joint conference on artificial intelligence, Hyderabad, pp 931–936
McCarthy J (1980) Circumscription – a form of non-monotonic reasoning. Artif Intell 13:27–39
Minton S (1990) Quantitative results concerning the utility of explanation-based learning. Artif Intell 42(2–3):363–391
Mitchell T (1997) Machine learning. McGraw-Hill, New York
Mitchell T, Keller R, Kedar-Cabelli S (1986) Explanation-based generalization: a unifying view. Mach Learn 1(1):47–80
Ourston D, Mooney RJ (1994) Theory refinement combining analytical and empirical methods. Artif Intell 66(2):273–309
Pazzani MJ, Kibler DF (1992) The utility of knowledge in inductive learning. Mach Learn 9:57–94
Russell SJ, Grosof BN (1987) A declarative approach to bias in concept learning. In: AAAI, Seattle, pp 505–510
Russell S, Norvig P (2003) Artificial intelligence: a modern approach, 2nd edn. Prentice-Hall, Englewood Cliffs
Sun Q, DeJong G (2005) Feature kernel functions: improving SVMs using high-level knowledge. In: CVPR (2), San Diego, pp 177–183
Thrun S, Mitchell TM (1993) Integrating inductive neural network learning and explanation-based learning. In: IJCAI'93: proceedings of the thirteenth international joint conference on artificial intelligence, Chambery, pp 930–936
Towell GG, Craven M, Shavlik JW (1991) Constructive induction in knowledge-based neural networks. In: proceedings of the eighth international conference on machine learning, Evanston, pp 213–217
Zelle JM, Mooney RJ (1993) Combining Foil and EBG to speed-up logic programs. In: IJCAI'93: proceedings of the thirteenth international joint conference on artificial intelligence, Chambery, pp 1106–1113

Explanation-Based Learning for Planning

Subbarao Kambhampati[1] and Sungwook Yoon[2]
[1]Arizona State University, Tempe, AZ, USA
[2]MapR, San Jose, CA, USA

Synonyms

Explanation-based generalization for planning; Speedup learning for planning

Definition

▸ Explanation-based learning (EBL) involves using prior knowledge to explain ("prove") why the training example has the label it is given and using this explanation to guide the learning. Since the explanations are often able to pinpoint the features of the example that justify its label, EBL techniques are able to get by with much fewer number of training examples. On the flip side, unlike general classification learners, EBL requires prior knowledge (aka "domain theory/model") in addition to labeled training examples – a requirement that is not easily met in some scenarios. Since many planning and problem-solving agents do start with declarative domain theories (consisting at least of descriptions of actions along with their preconditions and effects), EBL has been a popular learning technique for planning.

Dimensions of Variation

The application of EBL in planning varies along several dimensions: whether the learning was for improving the speed and quality of the underlying planner (Etzioni 1993; Kambhampati 1994; Kambhampati et al. 1996; Minton et al. 1989; Yoon et al. 2008) or acquire the domain model (Levine and DeJong 2006), whether it was done from successes (Kambhampati 1994; Yoon et al. 2008) or failures (Minton et al. 1989; Ihrig and Kambhampati 1997), whether the explanations were based on complete/correct (Minton et al. 1989; Kambhampati et al. 1996) or partial domain theories (Yoon et al.), whether learning is based on single (Kambhampati 1994; Kambhampati et al. 1996; Minton et al. 1989) or multiple examples (Flann and Dietterich 1989; Estlin and Mooney 1997) (where, in the latter case, inductive learning is used in conjunction with EBL), and finally whether the planner whose performance EBL aims to improve is a means-ends analysis one (Minton et al. 1989), partial-order planner (Estlin and Mooney 1997), or a heuristic search planner (Yoon et al.).

EBL techniques have been used in planning both to improve search and to reduce domain modeling burden (although the former has received more attention by far). In the former case, EBL is used to learn "control knowledge" to speed up the search process (Minton et al. 1989; Kambhampati et al. 1996) or to improve the quality of the solutions found by the search process (Estlin and Mooney 1997). In the latter case, EBL is used to develop domain models (e.g., action models) (Levine and DeJong 2006).

EBL for search improvement involves either remembering and reusing successful plans or learning search control rules to avoid failing search branches. Other variations include learning effective indexing of stored cases from retrieval failures (Ihrig and Kambhampati 1997) and learning "adjustments" to the default heuristic used by the underlying search.

Another important issue is the degree of completeness/correctness of the underlying background theory used to explain examples. If the theory is complete and correct, then learning is possible from a single example. This type of EBL has been called "analytical learning." When the theory is partial, EBL still is effective in narrowing down the set of potentially relevant features of the training example. These features can then be used within an inductive learner. Within planning, EBL has been used in the context of complete/correct as well as partial domain models.

A final dimension of variation that differentiated a large number of research efforts is the type of underlying planner. Initially, EBL was used on top of means-ends analysis planners (cf. PRODIGY, Minton et al. 1989). Later work focused on partial-order planners (e.g., Kambhampati et al. 1996; Estlin and Mooney 1997). More recently, the focus has been on forward search state-space planners (Yoon et al. 2008).

Learning from Success: Explanation-Based Generalization

When learning from successful cases (plans), the training examples comprise of successful plans, and the explanations involve proofs showing that the plan, as it is given, is able to support the goals. Only the parts of the plan that take part in this proof are relevant for justifying the success of the plan. The plan is thus "generalized" by removing extraneous actions that do not take part in the proof. Object identifiers and action orderings are also generalized as long as the generalization does not affect the proof of correctness (Kambhampati 1994). The output of the learning phase is thus a variablized plan containing a subset of the constraints (actions, orderings, object identity constraints) of the original plan. This is then typically indexed and used as a macro-operator to speed up later search.

For example, given a planning problem of starting with an initial state where five blocks, A, B, C, D, and E, are on the table, and the problem requires that in the goal state A must be on B and C must be on D and a plan P that is a sequence of actions *pickup A, stack A on B, pickup E, putdown E, Pickup C, stack C on D*, the explanation-based learner might output the generalization *do in any order* {*pickup x, stack x on y*} {*pick up z, stack z on w*} for the generalized goals *on* (x, y) *and on* (w, z), starting from a state where x, y, z, and w are all on the table and clear, and each of them denotes a distinct block.

One general class of such proof schema involves showing that every top-level goal of the planning problem as well as the precondition of every action is established and protected. Establishment requires that there is an action in the plan that gives that condition, and protection requires that once established, the condition is not deleted by any intervening action.

A crucial point is that the extent of generalization depends on the flexibility of the proof strategy used. Kambhampati and Kedar (1994) discuss a spectrum of generalization strategies associated with a spectrum of proof strategies, while Shavlik (1990) discusses how the number of actions in the plan can also be generalized.

Learning from Failure

When learning from the failure of a search branch, EBL starts by analyzing the plans at the failing nodes and constructing an explanation of failure. The failure explanation is just a subset of constraints in the plan at the current search node, which, in conjunction with domain theory, ensures that no successful solution can be reached by further refining this plan. The explanations can range from direct constraint inconsistencies (e.g., ordering cycles) to indirect violation of domain axioms (e.g., the plan requiring both clear(B) and On(A,B) to be satisfied at the same time point). The explanations at the leaf nodes are "regressed" over the decisions in the search tree to higher-level nodes to get explanations of (implicit) failures in these higher-level nodes. The search control rules can then essentially recommend pruning any search node which satisfies a failure explanation.

The deep affinity between EBL from search failures and the idea of ▶ nogood learning and dependency-directed backtracking in CSP is explored in Kambhampati (1998). As in dependency-directed backtracking, the more succinct the explanation, the higher the chance of learning effective control rules. Note that effectiveness here is defined in terms of the match costs involved in checking whether the rule is applicable and the search reductions provided when it is applicable. Significant work has been done to identify classes of failure explanation that are expected to lead to ineffective rules (Etzioni 1993). In contrast to CSP that has a finite depth search tree, one challenge in planning is that often an unpromising search node might not exhibit any direct failure with a succinct explanation and is abandoned by the search for heuristic reasons (such as the fact that the node crosses a depth limit threshold). Strategies for finding implicit explanations of failure (using domain axioms), as well as getting by with incomplete explanations of failure, are discussed in Kambhampati et al. (1996). EBL from failures has also been applied to retrieval (rather than search) failures. In this case, the failure of extending a plan retrieved from the library to solve a new problem is used to learn new indexing schemes that inhibit that case from being retrieved in such situations (Ihrig and Kambhampati 1997).

Learning Adjustments to Heuristics

Most recent work in planning has been in the context of heuristic search planners, where learning from failures does not work as well (since the heuristic search may change directions much before a given search branch ends in an explainable failure). One way of helping such planners is to improve their default heuristic (Yoon et al. 2008). Given a heuristic $h(s)$ that gives the heuristic estimate of state s, the aim in Yoon et al. is to learn an adjustment $\delta(s)$ that is added to $h(s)$ to get a better estimate of $h^*(s)$ – the true cost of state s. The system has access to actual plan traces (which can be obtained by having the underlying planner solve some problems from scratch). For each state s on the trace, we know the true distance of state s from the goal state, and we can also compute the $h(s)$ value with respect to the default heuristic. This gives the learner a set of training examples which are pairs of states and the adjustments they needed to make the default heuristic meet the true distance. In order to learn the $\delta(s)$ from this training data, we need to enumerate the features of state s that are relevant to it needing the specific adjustment. This is where EBL comes in. Specifically, one way of enumerating the relevant features is to explain why s has the default heuristic value. This, in turn, is done by taking the features of the relaxed plan for state s. Since the relaxed plan is a plan that assumes away all negative interactions between the actions, relaxed plan features can be seen as features of the explanation of the label for state s in terms of a *partial domain theory* (one which ignores all the deletes of all actions).

EBL from Incomplete Domain Theories

While most early efforts for speedup focused on complete and correct theories, several efforts also looked at speedup learning from incomplete theories. The so-called Lazy EBL approaches (Tadepalli 1989; Chien 1989) work by first constructing partial explanations and subsequently refining the over-general rules learned. Other approaches that use similar ideas outside planning include Flann and Dietterich (1989) and Cohen (1992). As we noted above, the work by Yoon et al. (2008) can also be seen as basing learning (in their case of adjustments to a default heuristic function) w.r.t. a partial domain theory.

EBL to Learn Domain Knowledge

Although most work in EBL for planning has been focused on speedup, there has also been some work aimed at learning domain knowledge (rather than control knowledge). Of particular interest is "operationalizing" a complex, if opaque, domain model by learning from it a simplified domain model that is adequate to efficiently solve an expected distribution of problems. The recent work by Levine and DeJong (2006) is an example of such an effort.

EBL and Knowledge-Level Learning

Although the focus of this article is on EBL as applied to planning, we need to foreground one general issue: whether EBL is capable of knowledge-level learning or not. A popular misconception of EBL is that since it depends on a complete and correct domain theory, no knowledge-level learning is possible, and speedup learning is the only possibility. (The origins of this misconception can be traced back to the very beginning. The two seminal articles on EBL in the very first issue of the Machine Learning journal differed profoundly in their interpretations of EBL. While Mitchell et al. (1986) assumed that EBL by default works with complete and correct theories (thus precluding any knowledge-level learning), Levine and DeJong (2006) provides a more general view of EBL that uses background knowledge – whether or not it is complete – to focus the generalization (and as such can be seen as a knowledge-based feature-selection step for a subsequent inductive learner).) As we noted at the outset however, EBL is not required to depend on complete and correct domain theories, and when it does not, knowledge-level learning is indeed possible.

Utility Problem and Its Nonexclusive Relation to EBL

As we saw above, much early work in EBL for planning focused on speedup for the underlying planner. Some of the knowledge learned for speedup – especially control rules and macro-operators – can also adversely affect the search by increasing either the search space size (macros) or per-node cost (matching control rules). Clearly, in order for the net effect to be positive, care needs to be exercised as to which control rules and/or macros are stored. This has been called the "utility problem" (Minton 1990), and significant attention has been paid to develop strategies that either dynamically evaluate the utility of the learned control knowledge (and forget useless rules) (Markovitch and Scott 1988; Minton 1990) or select the set of rules that best serve a given distribution of problem instances (Gratch et al. 1994).

Despite the prominent attention given to the utility problem, it is important to note the nonexclusive connection between EBL and utility problem. We note that *any* strategy that aims to provide/acquire control knowledge will suffer from the utility problem. For example, utility problem also holds for inductive learning techniques that were used to learn control knowledge (cf. Leckie and Zukerman 1993). In other words, it is not special to EBL but rather to the specific application task. We note that it is both possible to do speedup learning that is less susceptible to the utility problem (e.g., learn adjustments to heuristics, Yoon et al. 2008) and possible to use EBL for knowledge-level learning (Levine and DeJong 2006).

Current Status

EBL for planning was very much in vogue in the late 1980s and early 1990s. However, as the speed of the underlying planners increased drastically, the need for learning as a crutch to improve search efficiency reduced. There has however been a recent resurgence of interest, both in further speeding up the planners and in learning domain models. Starting 2008, there is a new track in the International Planning Competition devoted to learning methods for planning. In the first year, the emphasis was on speedup learning. ObtuseWedge, a system that uses EBL analysis to learn adjustments to the default heuristic, was among the winners of the track. The DARPA integrated learning initiative, and interest in model-lite planning have also brought focus back to EBL for planning – this time with partial domain theories.

Additional Reading

The tutorial (Yoon and Kambhampati 2007) provides an up-to-date and broader overview of learning techniques applied to planning and contains significant discussion of EBL techniques. The paper Zimmerman and Kambhampati (2003) provides a survey of machine learning techniques used in planning and includes a more comprehensive listing of research efforts that applied EBL in planning.

Cross-References

- ▶ Explanation-Based Learning
- ▶ Speedup Learning

Recommanded Reading

Bhatnagar N, Mostow J (1994) On-line learning from search failures. Mach Learn 15(1):69–117

Borrajo D, Veloso MM (1997) Lazy incremental learning of control knowledge for efficiently obtaining quality plans. Artif Intell Rev 11(1–5):371–405

Chien SA (1989) Using and refining simplifications: explanation-based learning of plans in intractable domains. In: IJCAI 1989, Detroit, pp 590–595

Cohen WW (1992) Abductive explanation-based learning: a solution to the multiple inconsistent explanation problem. Mach Learn 8:167–219

DeJong G, Mooney RJ (1986) Explanation-based learning: an alternative view. Mach Learn 1(2):145–176

Estlin TA, Mooney RJ (1997) Learning to improve both efficiency and quality of planning. In: IJCAI 1997, Nagoya, pp 1227–1233

Etzioni O (1993) A structural theory of explanation-based learning. Artif Intell 60(1):93–139

Flann NS, Dietterich TG (1989) A study of explanation-based methods for inductive learning. Mach Learn 4:187–226

Gratch J, Chien SA, DeJong G (1994) Improving learning performance through rational resource allocation. In: AAAI 1994, Seattle, pp 576–581

Ihrig LH, Kambhampati S (1997) Storing and indexing plan derivations through explanation-based analysis of retrieval failures. J Artif Intell Res 7:161–198

Kambhampati S (1994) A unified framework for explanation-based generalization of partially ordered and partially instantiated plans. Artif Intell 67(1):29–70

Kambhampati S (1998) On the relations between intelligent backtracking and failure-driven explanation-based learning in constraint satisfaction and planning. Artif Intell 105(1–2): 161–208

Kambhampati S, Katukam S, Qu Y (1996) Failure driven dynamic search control for partial order planners: an explanation based approach. Artif Intell 88(1–2):253–315

Leckie C, Zukerman I (1993) An inductive approach to learning search control rules for planning. In: IJCAI 1993, Chambéry, pp 1100–1105

Levine G, DeJong G (2006) Explanation-based acquisition of planning operators. In: ICAPS 2006, Cumbria, pp 152–161

Markovitch S, Scott PD (1988) The role of forgetting in learning. In: ML 1988, Ann Arbor, pp 459–465

Minton S (1990) Quantitative results concerning the utility of explanation-based learning. Artif Intell 42(2–3):363–391

Minton S, Carbonell JG, Knoblock CA, Kuokka D, Etzioni O, Gil Y (1989) Explanation-based learning: a problem solving perspective. Artif Intell 40(1–3):63–118

Mitchell TM, Keller RM, Kedar-Cabelli ST (1986) Explanation-based generalization: a unifying view. Mach Learn 1(1):47–80

Shavlik JW (1990) Acquiring recursive and iterative concepts with explanation-based learning. Mach Learn 5:39–40

Tadepalli P (1989) Lazy explanation based learning: a solution to the intractable theory problem. In: IJCAI 1989, Detroit, pp 694–700

Yoon S, Fern A, Givan R (2008) Learning control knowledge for forward search planning. J Mach Learn Res 9:683–718

Yoon S, Kambhampati S (2007) Learning for planning. Tutorial delivered at ICAPS 2007. http://rakaposhi.eas.asu.edu/learn-plan.html

Zimmerman T, Kambhampati S (2003) Learning-assisted automated planning: looking back, taking stock, going forward. AI Mag 24(2):73–96

F

F_1-Measure

The F_1-measure is used to evaluate the accuracy of predictions in two-class (binary) ▶ classification problems. It originates in the field of information retrieval and is often used to evaluate ▶ document classification models and algorithms. It is defined as the harmonic mean of ▶ precision (i.e., the ratio of ▶ true positives to all instances predicted as positive) and ▶ recall (i.e., the ratio of true positives to all instances that are actually positive). As such, it lies between precision and recall, but is closer to the smaller of these two values. Therefore a system with high F_1 has both good precision and good recall. The F_1-measure is a special case of the more general family of evaluation measures:

$$F_\beta = (\beta^2 + 1)\mathit{precisionrecall} / (\beta^2 \mathit{precision} + \mathit{recall})$$

Thus using $\beta >$ increases the influence of precision on the overall measure, while using $\beta < 1$ increases the influence of recall. Some authors use an alternative parameterization,

$$F_\alpha = 1/(\alpha/\mathit{precision} + (1t\alpha)/\mathit{recall})$$

which, however, leads to the same family of measures; conversion is possible via the relationship $\alpha = 1/(\beta^2 + 1)$.

False Negative

In a two-class problem, a ▶ classification model makes two types of error: ▶ false positives and false negatives. A **false negative** is an example of positive class that has been incorrectly classified as negative. See ▶ confusion matrix for a complete range of related terms.

False Positive

In a two-class problem, a ▶ classification model makes two types of error: false positives and ▶ false negatives. A **false positive** is an example of a negative class that has been incorrectly classified as positive. See ▶ confusion matrix for a complete range of related terms.

Feature

▶ Attribute

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

Feature Construction in Text Mining

Janez Brank[1], Dunja Mladenić[2], and
Marko Grobelnik[2]
[1]Jožef Stefan Insitute, Ljubljana, Slovenia
[2]Artificial Intelligence Laboratory, Jožef Stefan Insitute, Ljubljana, Slovenia

Synonyms

Feature generation in text mining

Definition

Feature construction in text mining consists of various techniques and approaches which convert textual data into a feature-based representation. Since traditional machine learning and data mining techniques are generally not designed to deal directly with textual data, feature construction is an important preliminary step in text mining, converting source documents into a representation that a data mining algorithm can then work with. Various kinds of feature construction approaches are used in text mining depending on the task that is being addressed, the data mining algorithms used, and the nature of the dataset in question.

Motivation and Background

Text mining is the use of machine learning and data mining techniques on textual data. This data consists of natural language documents that can be more or less structured, ranging from completely unstructured plain text to documents with various kinds of tags containing machine-readable semantic information. Furthermore, documents may sometimes contain hyperlinks that connect them into a graph. Since most traditional machine learning and data mining techniques are not directly equipped to deal with this kind of data, an important first step in text mining is to extract or construct features from the input documents, thereby obtaining a feature-based representation which is suitable for handling with machine learning and data mining algorithms. Thus, the task of feature construction in text mining is inextricably connected with text mining itself and has evolved alongside it. An important trend over the years has been the development of techniques that do not process each document in isolation but make use of a corpus of documents as a whole, possibly even involving external data or background knowledge in the process.

Documents and text data provide for valuable sources of information, and their growing availability in electronic form naturally led to application of different analytic methods. One of the common ways is to take a whole vocabulary of the natural language in which the text is written as a feature set, resulting in several tens of thousands of features. In a simple setting, each feature gives a count of the word occurrences in a document. In this way text of a document is represented as a vector of numbers. The representation of a particular document contains many zeros, as most of the words from the vocabulary do not occur in a particular document. In addition to the already mentioned two common specifics of text data, having a large number of features and a sparse data representation, it was observed that frequency of words in text in general follows Zipf's law – a small subset of words occur very frequently in texts, while a large number of words occur only rarely. Document classification takes these and some other data specifics into account when developing the appropriate classification methods.

Structure of Learning System

In a learning or mining system that deals with textual data, feature construction is usually one of the first steps that is often performed alongside typical preprocessing tasks such as data cleaning. A typical output of feature construction is feature vector representing the input documents; these vectors themselves then form the input for a machine learning or data mining algorithm. On the other hand, sometimes feature construction is more closely integrated into the learning algorithm itself, and sometimes it can be argued that the features themselves are the desired output that is the goal of the text mining task.

Solutions

At the lowest level, text is represented as a sequence of bytes or other elementary units of information. How these bytes are to be converted into a sequence of characters depends on the *character encoding* of the text. Many standard encodings exist, such as UTF-8, the ISO-8859 family, and so on. Often, all the texts that appear as input for a specific text mining task are in the same encoding, or if various encodings are used, they are specified clearly and explicitly (e.g., via the Content-Type header in the HTTP protocol), in which case the problem of conversion is straightforward. In the case of missing or faulty encoding information, various heuristics can be used to detect the encoding and convert the data to characters; it is best to think of this as a data cleaning and preprocessing step.

Word-Based Features

When we have our text represented as a sequence of characters, the usual next step is to convert it into a sequence of words. This is usually performed with heuristics which depend to some extent on the language and underlying character set; for the purposes of segmentation of text into words, a word is thought of as a sequence of alphabetic characters delimited by whitespace and/or punctuation. Some efforts to standardize word boundary detection in a way that would work reasonably well with a large set of natural languages have also been made (see, e.g., the Unicode Standard Annex #29, *Unicode Text Segmentation*). For many (but not all) text mining tasks, the distinction between upper- and lowercase (if it is present in the underlying natural language) is largely or entirely irrelevant; hence, all texts are often converted into lowercase at this point. Another frequently used preprocessing step is *stemming*, whereby each word is replaced by its stem (e.g., *walking* → *walk*). The details of stemming depend on the natural language involved; for English, a relatively simple set of heuristics such as Porter's stemmer is sufficient. Instead of stemming, where the ending is chopped off the word, one can apply a more sophisticated transformation referred to as lemmatization that replaces the word by its normalized form (lemma). Lemmatization is especially relevant for natural languages that have many different forms of the same word (e.g., several cases, gender influence on verb form, etc.). Efforts have also been made to discover stemming rules or lemmatization rules automatically using machine learning techniques (Plisson et al. 2008).

The individual words can themselves be thought of as features of the document. In the feature vector representation of a document d, the feature corresponding to the word w would tell something about the presence of the word w in this document: either the frequency (number of occurrences) of w in d, or a simple binary value (1 if present, 0 if absent), or it can further be modified by, e.g., the TF-IDF weighting. In this kind of representation, all information about the word order in the original document is lost; hence, it is referred to as the "bag-of-words" model. For many tasks, the loss of word order information is not critical, and the bag-of-words model is a staple of information retrieval, document classification, and many other text-related tasks. A downside of this approach (and many other word-based feature construction techniques) is that the resulting number of features can be very large (there are easily tens of thousands of different words in a mid-sized document corpus); see Feature Selection in Text Mining.

Clearly, ignoring the word order completely can sometimes lead to the loss of valuable information. Multi-word phrases sometimes have a meaning that is not adequately covered by the individual words of the phrase (e.g., proper names, technical terms, etc.). Various ways of creating multi-word features have been considered. Let d be a document consisting of the sequence of words $(w_1, w_2, \ldots, w_m)$ (note that this sequence might already be an output of some preprocessing operations, e.g., the removal of stopwords and of very infrequent words). Then an *n-gram* is defined as a sequence of n adjacent words from the document, i.e., $(w_i, w_{i+1}, \ldots, w_{i+n-1})$. We can use n-grams as features in the same way as individual words, and indeed a typical approach is to use n-grams for all values of n from 1 to a certain upper limit (e.g., 5). Many of the resulting n-grams will be incidental and irrelevant, but some of them may be valuable and informative phrases; whether the text mining algorithm will be able to profit from them depends a lot on the algorithm used, and feature selection might be even more necessary than in the case of individual words. A related problem is the explosion of the number of features; if the number of different words in a corpus grows approximately with the square root of the length of the corpus (Heaps' law), the number of different n-grams is more likely to grow nearly linearly with the length of the corpus. The use of n-grams as features has been found to be beneficial, e.g., for the classification of very short documents (Mladenić and Grobelnik 2003).

Further generalization of n-grams is possible by removing the requirement that the words of the n-gram must appear adjacently; we can allow them to be separated by other words. The weight of an occurrence of the n-gram is often defined as decreasing exponentially with the number of intervening separator words. Another direction of generalizing n-gram is to ignore the order of words within the n-gram; in effect one treats n-grams as bags (multisets) instead of sequences. This results in features sometimes called *loose phrases* or *proximity features* (i.e., every bag of words up to a certain size, occurring in sufficiently close proximity to each other, is considered to be a feature). These generalizations greatly increase the feature space as well as the number of features present in any individual document, so the risk of computational intractability is greatly increased; this can sometimes be alleviated through the use of kernels (see below).

Character-Based Features

Instead of treating the text as a sequences of words, we might choose to treat it as a sequence of characters. A sequence of n characters is also known as an n-graph. We can use n-graphs as features in the representation of text in a way analogous to the use of n-grams in the previous subsection. The weight of the feature corresponding to a particular n-graph in the feature vector of a particular document d will typically depend on the number of occurrences of that n-graph in the text of d. Sometimes noncontiguous occurrences of the n-graph are also counted (i.e., occurrences where characters from the n-graph are separated by one or more other characters), although with a lower weight; this is can be done very elegantly with kernel methods (see below). Feature selection and TF-IDF style weighting schemes can also be used as in the case of n-grams. Whether an n-graph-based representation offers any benefits compared to an n-gram-based one depends largely on the dataset and task in question. For example, the classification of English documents and the usefulness of n-graphs have been found to be dubious, but they can be beneficial in highly agglutinative languages where an individual word can consist of many morphemes, and it is not really useful to treat a whole word as an individual unit of information (as would be the case in a word-based feature representation). In effect, the use of n-graphs provides the learner with cheap access to the sort of information that would otherwise require more sophisticated NLP technologies (stemming, parsing, morpheme analysis, etc.); the downside is that a lot of the n-graph features are merely noise (Lodhi et al. 2002). For some application, word suffixes can be particularly useful features, e.g., to learn lemmatization rules (Mladenić 2002; Plisson et al. 2008).

Kernel Methods

Let φ be a function which assigns, to a given document d, a feature vector $\varphi(d)$ from some feature space F. Assume furthermore that a dot product (a.k.a. inner product) is defined over F, denoted by $\langle \cdot, \cdot \rangle_F$. Then the function K defined by $K(d_1, d_2) = \langle \varphi(d_1), \varphi(d_2) \rangle_F$ is called a kernel function. It turns out that many machine learning and data mining methods can be described in a way such that the only operation they need to do with the data is to compute dot products of their feature vectors; in other words, they only require us to be able to compute the kernel function over our documents. These approaches are collectively known as *kernel methods*; a well-known example of this is the support vector machine (SVM) method for supervised learning, but the same principle can be used in clustering as well. An important advantage of this approach is that it is often possible to compute the kernel function K directly from the documents $d_{1,2}$ without explicitly generating the feature vectors $\varphi(d_{1,2})$. This is especially valuable if the feature space is untractably large. Several families of kernel functions for textual data have been described in the literature, corresponding to various kinds of n-graph and n-gram-based features (Brank 2006; Lodhi et al. 2002).

Linear Algebra Methods

Assume that a corpus of n documents have already been represented by d-dimensional real feature vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in R^d$. If we select some direction $\mathbf{y} \in R^d$ and project a vector $\mathbf{x}_i$ in this direction, the resulting value $\mathbf{y}^T \mathbf{x}_i / ||\mathbf{y}||$ is in effect a new feature describing the document i. In other words, we have constructed a new feature as a linear combination of the existing features. This leads to the question of how to select one or more suitable directions $\mathbf{y}$; various techniques from linear algebra and statistics have been proposed for this.

A well-known example of this is principal component analysis (PCA) in which one or more new coordinate axes $\mathbf{y}$ are selected in such a way that the variance of the original vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in the directions of the new coordinate axes is maximized. As it turns out, this problem is equivalent to computing the principal eigenvectors of the covariance matrix of the original dataset.

Another technique of this sort is latent semantic indexing (LSI) (Deerwester et al. 1990). Let X be a $d \times n$ matrix with $\mathbf{x}_1, \ldots, \mathbf{x}_n$ as its columns (a.k.a. the *term-document matrix*). LSI uses singular value decomposition (SVD) to express X as the product of three matrices, $T \cdot S \cdot D$, where T is a $d \times r$ orthonormal matrix, D is a $r \times n$ orthonormal matrix, and S is a $r \times r$ diagonal matrix containing the singular values of X. Here, r denotes the rank of the original matrix X. Let $T^{(m)}$ be the matrix consisting of the left m columns of T, let $D^{(m)}$ be the matrix consisting of the top m rows of D, and let $S^{(m)}$ be the top left $m \times m$ submatrix of S. Then it turns out that $X^{(m)} = T^{(m)} S^{(m)} D^{(m)}$ is the best rank-m approximation of the original X (best in the sense of minimizing the Frobenius norm of $X - X^{(m)}$). Thus, the i-th column of $D^{(m)}$ can be seen as a vector of m new features representing the i-th document of our original dataset, and the product $T^{(m)} S^{(m)}$ can be seen as a set of m new coordinate axes. The new feature vectors (columns of $D^{(m)}$) can be used instead of the original vectors $\mathbf{x}_i$.

Canonical correlation analysis (CCA): Sometimes several vector representations are available for the same document d_i, for example, we might have the same text in two different languages, giving rise to two feature vectors, e.g., $\mathbf{x}_i \in R^d$ and $\mathbf{y}_i \in R^{d'}$. Given such a "parallel corpus" of pairs $(\mathbf{x}_i, \mathbf{y}_i), i = 1 \ldots n$, it is sometimes desirable to convert both types of representations to a "common denominator." In other words, we want to find a set of r new coordinate axes in $\mathbf{x}$-space (say the columns of $U \in R^{d \times r}$) and a set of r new coordinate axes in $\mathbf{y}$-space (say the columns of $V \in R^{d\prime \times r}$) such that the j-th column of U has a similar role in $\mathbf{x}$-space as the j-th column of V has in $\mathbf{y}$-space, for all j. This can be formulated as an optimization problem: find U and V such that the correlation between U^T $\mathbf{x}_i$ and $V^T \mathbf{y}_i$ (i.e., the projections of $\mathbf{x}_i$ and $\mathbf{y}_i$ onto the new sets of axes) is maximized. Once we have suitable matrices U and V , we can convert any feature vector from the original $\mathbf{x}$-space or $\mathbf{y}$-space into a common new r-dimensional space. This makes it easier to deal with multilingual corpora, allowing us, e.g., to retrieve documents in language $\mathbf{x}$ as a response to a query in language $\mathbf{y}$ or vice versa. The same techniques are applicable in multimodal scenarios (i.e., $\mathbf{x}_i$ and $\mathbf{y}_i$ can be any two representations of the same instance d_i from two substantially different perspectives, not necessarily textual). This method is often used in combination with kernels, in which case it is known as *kernel* canonical correlation analysis (KCCA) (Hardoon et al. 2004).

Nonlinear Methods

Powerful feature representations can also be obtained by statistical methods. For example, probabilistic latent semantic analysis (PLSA) (Hoffmann 1999) is an unsupervised approach that models a corpus of documents as if it was generated by a mixture of latent topics, with each topic being represented as a probability distribution over words. The model consists of topic probabilities $P(z|d)$ for each latent topic z and each document d and of word probabilities $P(w|z)$ for each word and topic. One of the downsides of PLSA is that it cannot be readily extended to model probabilities of documents that were not seen during training; this problem is addressed by approaches such as latent Dirichlet allocation (LDA) (Blei et al. 2003), in which the mixture of topics in each document is modeled as a random variable sampled from a Dirichlet distribution. In both cases, the mixture of topics $\langle P(z|d)\rangle_z$ can be interpreted as a new feature vector representation of the document d, and likewise the conditional probabilities $.\langle P(w|z)\rangle_z$ can be interpreted as a new feature vector representation of the term w.

Another important family of nonlinear feature construction methods are *deep learning methods*, which are based on training a multilevel neural network model that includes at least one hidden layer. Traditionally one would be interested in the outputs of the final output layer of the network, which is supposed to be solving whatever learning task the network was originally trained for. However, in deep learning, one discards the output layers and instead uses the outputs of the hidden layer as a new feature vector representation of the input document (or a word, n-gram, etc.). A recent example of such a representation is *word2vec* (Mikolov et al. 2013).

Miscellaneous

There are many other ways to extract or construct features from text, depending on the use that the features are intended for. For example, a *dual representation* of a corpus may be considered, in which features are used to represent terms and not documents. The feature vector for a term t contains one feature for each document, and its value is related to the frequency of t in that document. This representation can be used to analyze which words co-occur frequently and may therefore be related in meaning. Feature construction can also utilize methods from *information extraction*, such as identifying various kinds of named entities (names of persons, places, organizations, etc.) or other interesting bits of information and introducing features which indicate the presence of particular names or other tagged entities in the document.

Cross-References

▶ Deep Learning
▶ Document Classification
▶ Feature Selection in Text Mining
▶ Kernel Methods
▶ Support Vector Machines
▶ Text Mining

Recommended Reading

Blei DM, Ng AY, Jordan MI (2003) Latent dirichlet allocation. J Mach Learn Res 3:993–1022

Brank J (2006) Loose phrase string kernels. In: Proceedings of SiKDD, Jozef Stefan Institute, Ljubljana

Deerwester S, Dumais ST, Furnas GW, Landauer TK, Harshman R (1990) Indexing by latent semantic analysis. J Am Soc Inf Sci 41:391–407

Hardoon DR, Szedmak SR, Shawe-Taylor JR (2004) Canonical correlation analysis: an overview with application to learning methods. Neural Comput 16(12):2639–2664

Hoffmann T (1999) Probabilistic latent semantic indexing. In: Proceedings of the 22nd SIGIR conference, Berkeley, pp 50–57

Lodhi H, Saunders C, Shawe-Taylor J, Cristianini N, Watkins C (2002) Text classification using string kernels. J Mach Learn Res 2:419–444

Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. In: International conference on learning representations, Scottsdale

Mladenić D (2002) Learning word normalization using word suffix and context from unlabeled data. In: Proceedings of the 19th ICML, Sydney, vol 1(8), pp 427–434

Mladenić D, Grobelnik M (2003) Feature selection on hierarchy of web documents. Decis Support Syst 35(1):45–87

Plisson J, Lavrač N, Mladenić D, Erjavec T (2008) Ripple down rule learning for automated word lemmatization. AI Commun 21(1):15–26

Shawe-Taylor J, Cristianini N (2004) Kernel methods for pattern analysis. Cambridge University Press, Cambridge

Feature Generation in Text Mining

▶ Feature Construction in Text Mining

Feature Projection

▶ Dimensionality Reduction

Feature Selection

Suhang Wang[1], Jiliang Tang[2], and Huan Liu[1]
[1]Arizona State University, Tempe, AZ, USA
[2]Michigan State University, East Lansing, MI, USA

Abstract

Data dimensionality is growing rapidly, which poses challenges to the vast majority of existing mining and learning algorithms, such as the curse of dimensionality, large storage requirement, and high computational cost. Feature selection has been proven to be an effective and efficient way to prepare high-dimensional data for data mining and machine learning. The recent emergence of novel techniques and new types of data and features not only advances existing feature selection research but also evolves feature selection continually, becoming applicable to a broader range of applications. In this entry, we aim to provide a basic introduction to feature selection including basic concepts, classifications of existing systems, recent development, and applications.

Synonyms

Attribute selection; Feature subset selection; Feature weighting

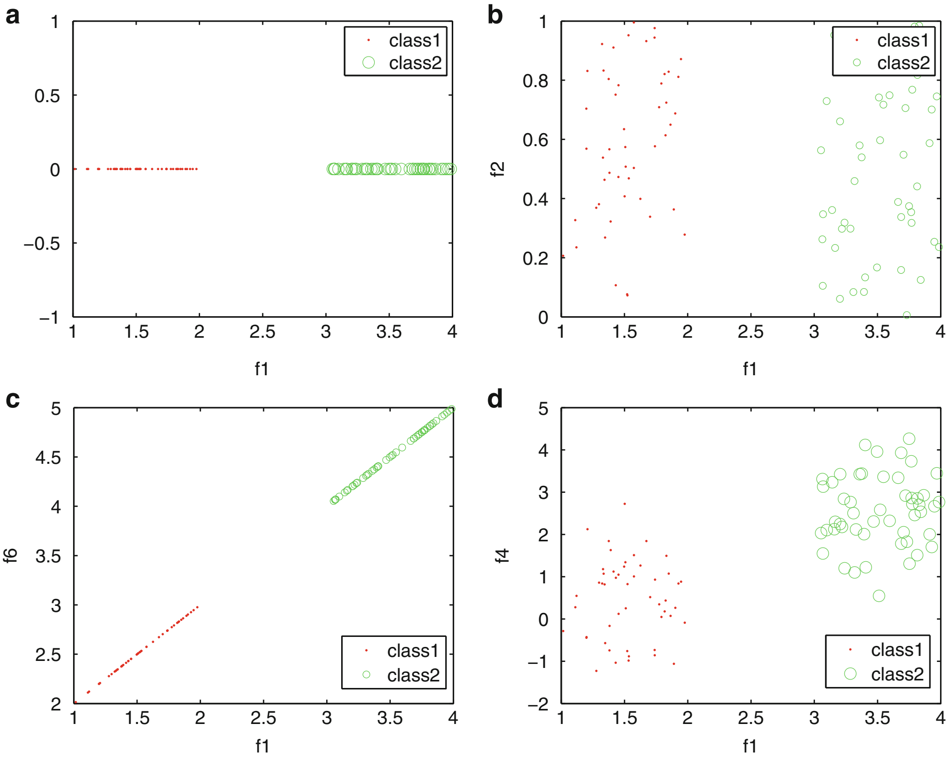
Definition (or Synopsis)

Feature selection, as a dimensionality reduction technique, aims to choose a small subset of the relevant features from the original ones by removing irrelevant, redundant, or noisy features. Feature selection usually leads to better learning performance, i.e., higher learning accuracy, lower computational cost, and better model interpretability.

Generally speaking, irrelevant features are features that cannot help discriminate samples from different classes(supervised) or clusters(unsupervised). Removing irrelevant features will not affect learning performance. In fact, the removal of irrelevant features may help learn a better model, as irrelevant features may confuse the learning system and cause memory and computation inefficiency. For example, in Fig. 1a, f_1 is a relevant feature because f_1 can discriminate class1 and class2. In Fig. 1b, f_2 is a redundant feature because f_2 cannot distinguish points from class1 and class2. Removal of f_2 doesn't affect the ability of f_1 to distinguish samples from class1 and class2.

A redundant feature is a feature that implies the copresence of another feature. Individually, each redundant feature is relevant, but removal of one of them will not affect the learning performance. For example, in Fig. 1c, f_1 and f_6 are strongly correlated. f_6 is a relevant feature itself. However, when f_1 is selected first, the later appearance of f_6 doesn't provide additional information. Instead, it adds more memory and computational requirement to learn the classification model.

A noisy feature is a type of relevant feature. However, due to the noise introduced during the data collection process or because of the nature of this feature, a noisy feature may not be so relevant to the learning or mining task. As shown in Fig. 1d, f_4 is a noisy feature. It can



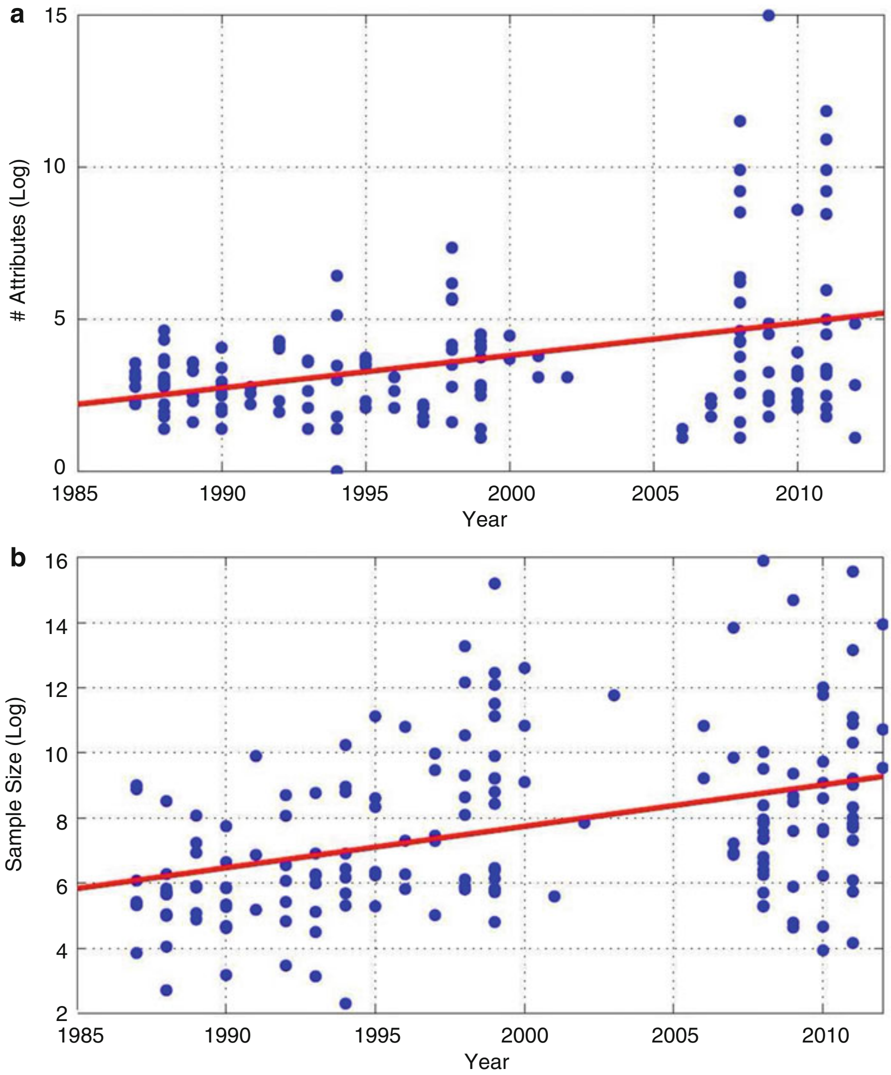
Feature Selection, Fig. 1 A toy example to illustrate the concept of irrelevant, redundant, and noisy features. $f1$ is a relevant feature and can discriminate class1 and class2. $f2$ is an irrelevant feature. Removal of $f2$ will not affect the learning performance. $f4$ is a noisy feature. The presence of noisy features may degenerate the learning performance. $f6$ is a redundant feature when $f1$ is present. If $f1$ is selected, removal of $f6$ will not affect the learning performance. (**a**) Relevant feature. (**b**) Irrelevant feature. (**c**) Redundant feature. (**d**) Noisy feature

discriminate a part of the points from the two classes and may confuse the learning model for the overlapping points (Noisy features are very subtle. One feature may be a noisy feature itself. However, in some cases, when two or more noisy features can complement each other to distinguish samples from different classes, they may be selected together to benefit the learning model.)

Motivation and Background

In many real-world applications, such as data mining, machine learning, computer vision, and bioinformatics, we need to deal with high-dimensional data. In the past 30 years, the dimensionality of the data involved in these areas has increased explosively. The growth of the number of attributes in the UCI machine learning repository is shown in Fig. 2a. In addition, the number of samples also increases explosively. The growth of the number of samples in the UCI machine learning repository is shown in Fig. 2b. The huge number of high-dimensional data has presented serious challenges to existing learning methods. First, due to the large number of features and relatively small number of training samples, a learning model tends to overfit, and their learning performance degenerates. Data with high dimensionality not only degenerates many algorithms' performance due to the curse of dimensionality and the existence of irrelevant, redundant, and noisy dimensions, it also significantly increases the time and memory requirement of the algorithms. Second, storing and processing such amounts of high-dimensional data become a challenge.

Dimensionality reduction is one of the most popular techniques to reduce dimensionality



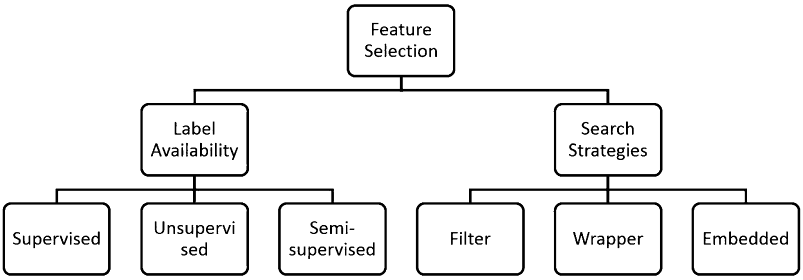
Feature Selection, Fig. 2 Growth of the number of features and the number of samples in the UCI ML repository. (**a**) UCI ML repository number of attribute growth. (**b**) UCI ML repository number of sample growth

and can be categorized into feature extraction and feature selection. Both feature extraction and feature selection are capable of improving performance, lowering computational complexity, building better generalization models, and decreasing required storage. Feature extraction maps the original feature space to a new feature space with lower dimensionality by combining the original feature space. Therefore, further analysis of new features is problematic since there is no physical meaning for the transformed features obtained from feature extraction. In contrast, feature selection selects a subset of features from the original feature set. Therefore, feature selection keeps the actual meaning of each selected feature, which makes it superior in terms of feature readability and interpretability.

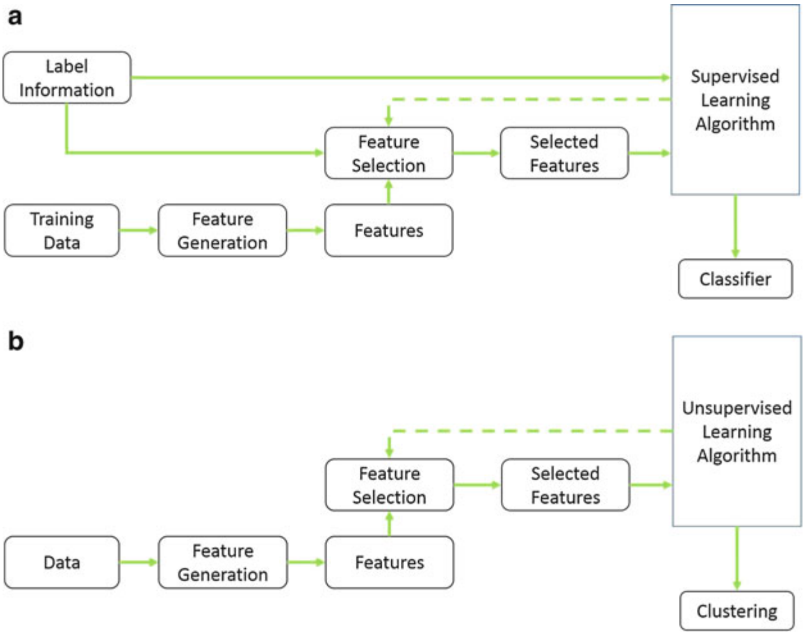
Structure of the Learning System

From the perspective of label availability, feature selection methods can be broadly classified into supervised, unsupervised, and semi-supervised

Feature Selection, Fig. 3 Feature selection categories



Feature Selection, Fig. 4 General frameworks of supervised and unsupervised feature selection. (**a**) A general framework of supervised feature selection. (**b**) A general framework of unsupervised feature selection



methods. In terms of different selection strategies, feature selection can be categorized as filter, wrapper, and embedded models. Figure 3 shows the classification of feature selection methods.

Supervised feature selection is usually used for classification tasks. The availability of the class labels allows supervised feature selection algorithms to effectively select discriminative features to distinguish samples from different classes. A general framework of supervised feature selection is shown in Fig. 4a. Features are first generated from training data. Instead of using all the data to train the supervised learning model, supervised feature selection will first select a subset of features and then process the data with the selected features to the learning model. The feature selection phase will use the label information and the characteristics of the data, such as information gain or Gini index, to select relevant features. The final selected features, as well as with the label information, are used to train a classifier, which can be used for prediction.

Unsupervised feature selection is usually used for clustering tasks. A general framework of unsupervised feature selection is described in Fig. 4b, which is very similar to supervised feature selection, except that there's no label information involved in the feature selection phase and the model learning phase. Without label information to define feature relevance, unsupervised feature selection relies on another alternative criterion during the feature selection phase. One commonly used criterion chooses features that can best preserve the manifold structure of the original data. Another frequently used method is to seek cluster indicators through clustering algorithms and then transform the unsupervised feature selection into a supervised framework. There are two different ways to use this method. One way is to seek cluster indicators and simultaneously perform the supervised feature selection within one unified framework. The other way is to first seek cluster indicators, then to perform feature selection to remove or select certain features, and finally to repeat these two steps iteratively until certain criterion is met. In addition, certain supervised feature selection criterion can still be used with some modification.

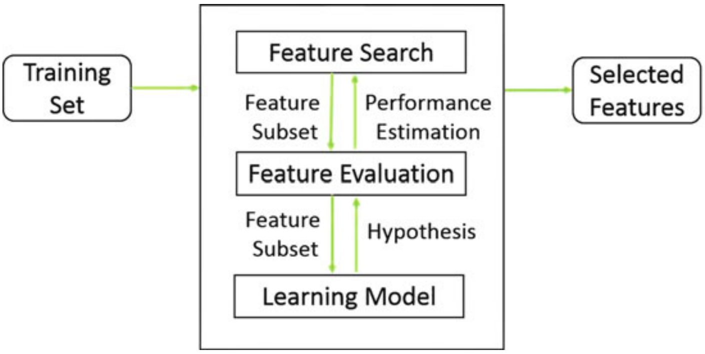
Semi-supervised feature selection is usually used when a small portion of the data is labeled. When such data is given to perform feature selection, both supervised and unsupervised feature selection might not be the best choice. Supervised feature selection might not be able to select relevant features because the labeled data is insufficient to represent the distribution of the features. Unsupervised feature selection will not use the label information, while label information can give some discriminative information to select relevant features. Semi-supervised feature selection, which takes advantage of both labeled data and unlabeled data, is a better choice to handle partially labeled data. The general framework of semi-supervised feature selection is the same as that of supervised feature selection, except that data is partially labeled. Most of the existing semi-supervised feature selection algorithms rely on the construction of the similarity matrix and select features that best fit the similarity matrix. Both the label information and the similarity measure of the labeled and unlabeled data are used to construct the similarity matrix so that label information can provide discriminative information to select relevant features, while unlabeled data provide complementary information.

Filter Models For filter models, features are selected based on the characteristics of the data without utilizing learning algorithms. This approach is very efficient. However, it doesn't consider the bias and heuristics of the learning algorithms. Thus, it may miss features that are relevant for the target learning algorithm. A filter algorithm usually consists of two steps. In the first step, features are ranked based on certain criterion. In the second step, features with the highest rankings are chosen. A lot of ranking criteria, which measures different characteristics of the features, are proposed: the ability to effectively separate samples from different classes by considering between class variance and within class variance, the dependence between the feature and the class label, the correlation between feature-class and feature-feature, the ability to preserve the manifold structure, the mutual information between the features, and so on.

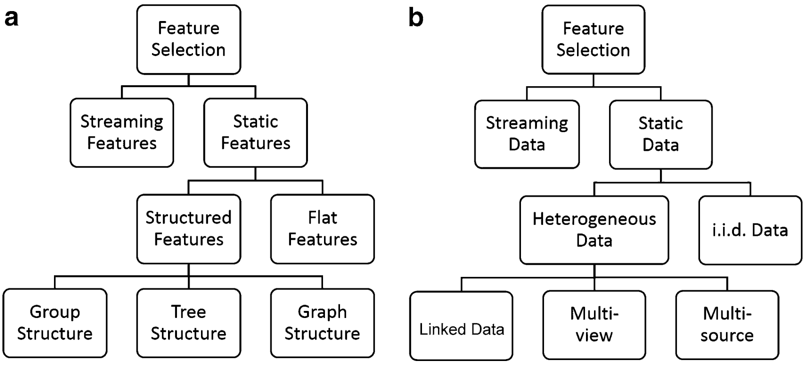
Wrapper Models The major disadvantage of the filter approach is that it totally ignores the effects of the selected feature subset on the performance of the clustering or classification algorithm. The optimal feature subset should depend on the specific biases and heuristics of the learning algorithms. Based on this assumption, wrapper models use a specific learning algorithm to evaluate the quality of the selected features. Given a predefined learning algorithm, a general framework of the wrapper model is shown in Fig. 5. The feature search component will produce a set of features based on certain search strategies. The feature evaluation component will then use the predefined learning algorithm to evaluate the performance, which will be returned to the feature search component for the next iteration of feature subset selection. The feature set with the best performance will be chosen as the final set. The search space for m features is $O(2^m)$. To avoid exhaustive search, a wide range of search strategies can be used, including hill-climbing, best-first, branch-and-bound, and genetic algorithms.

Embedded Models Filter models are computationally efficient, but totally ignore the biases of the learning algorithm. Compared with filter models, wrapper models obtain better predictive accuracy estimates, since they take into account the biases of the learning algorithms. However, wrapper models are very computationally expensive. Embedded models are a tradeoff between the two models by embedding the feature selection into the model construction. Thus, embedded models take advantage of both filter models and wrapper models: (1) they are

Feature Selection, Fig. 5 A general framework of wrapper models



Feature Selection, Fig. 6 Classification of recent development of feature selection from feature perspective and data perspective



far less computationally intensive than wrapper methods, since they don't need to run the learning models many times to evaluate the features, and (2) they include the interaction with the learning model. The biggest difference between wrapper models and embedded models is that wrapper models first train learning models using the candidate features and then perform feature selection by evaluating features using the learning model, while embedded models select features during the process of model construction to perform feature selection without further evaluation of the features.

Recent Developments

The recent emergence of new machine learning algorithms, such as sparse learning, and new types of data, such as social media data, has accelerated the evolution of feature selection. In this section, we will discuss recent developments of feature selection from both feature and data perspectives.

From the feature perspective, features can be categorized as static and streaming features, as shown in Fig. 6a. Static features can be further categorized as flat features and structured features. The recent development of feature selection from the feature perspective mainly focuses on streaming and structure features.

Usually we assume that all features are known in advance. These features are designated as static features. In some scenarios, new features are sequentially presented to the learning algorithm. For example, Twitter produces more than 250 millions tweets per day, and many new words (features) are generated, such as abbreviations. In these scenarios, the candidate features are generated dynamically, and the size of features is unknown. These features are usually named as streaming features, and feature selection for streaming features is called streaming feature selection. For flat features, we assume that features are independent. However, in many real-world applications, features may exhibit certain intrinsic structures, such as overlapping groups, trees, and graph structures. For example, in speed and signal processing, different frequency bands can be represented by groups. Figure 6a shows the classification of structured features. Incorporating knowledge about feature structures may significantly improve the performance of learning models and help select important features. Feature selection algorithms for the structured features usually use the recently developed sparse learning techniques such as group lasso and tree-guided lasso.

From the data perspective, data can be categorized as streaming data and static data as shown in Fig. 6b. Static data can be further categorized as independent identically distributed (i.i.d.) data and heterogeneous data. The recent development of feature selection from the data perspective is mainly concentrated on streaming and heterogeneous data.

Similar to streaming features, streaming data comes sequentially. Online streaming feature selection is proposed to deal with streaming data. When new data instances come, an online feature selection algorithm needs to determine (1) whether adding the newly generated features from the coming data to the currently selected features and (2) whether removing features from the set of currently selected features ID. Traditional data is usually assumed to be i.i.d. data, such as text and gene data. However, heterogeneous data, such as linked data, apparently contradicts this assumption. For example, linked data is inherently not i.i.d., since instances are linked and correlated. New types of data cultivate new types of feature selection algorithms correspondingly, such as feature selection for linked data and multi-view and multisource feature selection.

Applications

High-dimensional data is very ubiquitous in the real world, which makes feature selection a very popular and practical preprocessing technique for various real-world applications, such as text categorization, remote sensing, image retrieval, microarray analysis, mass spectrum analysis, sequence analysis, and so on.

Text Clustering The task of text clustering is to group similar documents together. In text clustering, a text or document is always represented as a bag of words, which causes high-dimensional feature space and sparse representation. Obviously, a single document has a sparse vector over the set of all terms. The performance of clustering algorithms degrades dramatically due to high dimensionality and data sparseness. Therefore, in practice, feature selection is a very important step to reduce the feature space in text clustering.

Genomic Microarray Data Microarray data is usually short and fat data – high dimensionality with a small sample size, which poses a great challenge for computational techniques. Their dimensionality can be up to tens of thousands of genes, while their sample sizes can only be several hundreds. Furthermore, additional experimental complications like noise and variability render the analysis of microarray data an exciting domain. Because of these issues, various feature selection algorithms are adopted to reduce the dimensionality and remove noise in microarray data analysis.

Hyperspectral Image Classification Hyperspectral sensors record the reflectance from the Earth's surface over the full range of solar wavelengths with high spectral resolution, which results in high-dimensional data that contains rich information for a wide range of applications. However, this high-dimensional data contains many irrelevant, noisy, and redundant features that are not important, useful, or desirable for specific tasks. Feature selection is a critical preprocessing step to reduce computational cost for hyperspectral data classification by selecting relevant features.

Sequence Analysis In bioinformatics, sequence analysis is a very important process to understand a sequence's features, functions, structure, or evolution. In addition to basic features that represent nucleotide or amino acids at each position in a sequence, many other features, such as k-mer patterns, can be derived. By varying the pattern length k, the number of features grows exponentially. However, many of these features are irrelevant or redundant; thus, feature selection techniques are applied to select a relevant feature subset and essential for sequence analysis.

Open Problems

Scalability With the rapid growth of dataset size, the scalability of current feature selection algorithms may be a big issue, especially for online classifiers. Large data cannot be loaded to the memory with a single scan. However, full dimensionality data must be scanned for some feature selection. Usually, they require a sufficient number of samples to obtain statistically significant result. It is very difficult to observe the feature relevance score without considering the density around each sample. Therefore, scalability is a big issue.

Stability Feature selection algorithms are often evaluated through classification accuracy or clustering accuracy. However, the stability of algorithms is also an important consideration when developing feature selection methods. For example, when feature selection is applied on gene data, the domain experts would like to see the same or at least similar sets of genes selected after each time they obtain new samples with a small amount of perturbation. Otherwise, they will not trust the algorithm. However, well-known feature selection methods, especially unsupervised feature selection algorithms, can select features with low stability after perturbation is introduced to the training data. Developing algorithms of feature selection with high accuracy and stability is still an open problem.

Parameter Selection In feature selection, we usually need to specify the number of features to select. However, the optimal number of features for the dataset is unknown. If the number of selected features is too few, the performance will be degenerated, since some relevant features are eliminated. If the number of selected features is too large, the performance may also not be very good since some noisy, irrelevant, or redundant features are selected to confuse the learning model. In practice, we would grid search the number of features in a range and pick the one that has relatively better performance on learning models, which is computationally expensive. In particular, for supervised feature selection, cross validation can be used to search the number of features to select. How to automatically determine the best number of selected features remains an open problem.

For many unsupervised feature selection methods, in addition to choosing the optimal number of features, we also need to specify the number of clusters. Since there is no label information and we have limited knowledge about each domain, the actual number of clusters in the data is usually unknown and not well defined. The number of clusters specified by users will result in selecting different feature subsets by the unsupervised feature selection algorithm. How to choose the number of clusters for unsupervised feature selection is an open problem.

Cross-References

▸ Classification
▸ Clustering
▸ Dimensionality Reduction
▸ Evolutionary Feature Selection

Recommended Reading

Alelyani S, Tang J, Liu H (2013) Feature selection for clustering: a review. In: Aggarwal CC (ed) Data clustering: algorithms and applications, vol 29. CRC Press, Hoboken

Dy JG, Brodley CE (2004) Feature selection for unsupervised learning. J Mach Learn Res 5:845–889

Guyon I, Elisseeff A (2003) An introduction to variable and feature selection. J Mach Learn Res 3: 1157–1182

Jain A, Zongker D (1997) Feature selection: evaluation, application, and small sample performance. IEEE Trans Pattern Anal Mach Intell 19(2):153–158

Kohavi R, John GH (1997) Wrappers for feature subset selection. Artif Intell 97(1):273–324

Koller D, Sahami M (1996) Toward optimal feature selection. Technical report, Stanford InfoLab

Li, J, Cheng K, Wang S, Morstatter F, Trevino R P, Tang J, Liu H (2016) Feature Selection: A Data Perspective. arXiv preprint 1601.07996

Liu H, Motoda H (2007) Computational methods of feature selection. CRC Press, New York
Liu H, Yu L (2005) Toward integrating feature selection algorithms for classification and clustering. IEEE Trans Knowl Data Eng 17(4):491–502
Liu H, Motoda H, Setiono R, Zhao Z (2010) Feature selection: an ever evolving frontier in data mining. In: FSDM, Hyderabad, pp 4–13
Saeys Y, Inza I, Larrañaga P (2007) A review of feature selection techniques in bioinformatics. Bioinformatics 23(19):2507–2517
Tang J, Liu H (2012) Feature selection with linked data in social media. In: SDM, Anaheim. SIAM, pp 118–128
Tang J, Alelyani S, Liu H (2014) Feature selection for classification: a review. In: Aggarwal CC (ed) Data classification: algorithms and applications. Chapman & Hall/CRC, Boca Raton, p 37
Wu X, Yu K, Ding W, Wang H, Zhu X (2013) Online feature selection with streaming features. IEEE Trans Pattern Anal Mach Intell 35(5):1178–1192
Zhao ZA, Liu H (2011) Spectral feature selection for data mining. Chapman & Hall/CRC, Boca Raton
Zhao Z, Morstatter F, Sharma S, Alelyani S, Anand A, Liu H (2010) Advancing feature selection research. ASU feature selection repository, 1–28

Feature Selection in Text Mining

Dunja Mladenić
Artificial Intelligence Laboratory, Jožef Stefan Insitute, Ljubljana, Slovenia

Abstract

Feature selection is commonly used when we are dealing with a large number of features or potentially noisy or redundant features. By selecting only some of the available features we hope to simplify the data representation, reduce the needed time and/or space for data processing, in some cases decrease the cost of collecting the feature values and, in some situations also to improve performance of the data modeling. Even though some approaches to feature selection are commonly applicable also in text mining, there are some specifics worth checking in tuning the existing methods or proposing some new methods for feature selection.

Synonyms

Dimensionality reduction on text via feature selection

Definition

The term *feature selection* is used in machine learning for the process of selecting a subset of features (dimensions) used to represent the data (see ▸ Feature Selection and ▸ Dimensionality Reduction). Feature selection can be seen as a part of data pre-processing potentially followed or coupled with ▸ feature construction (see ▸ Feature Construction in Text Mining), but can also be coupled with the learning phase if embedded in the learning algorithm. An assumption of feature selection is that we have defined an original feature space that can be used to represent the data, and our goal is to reduce its dimensionality by selecting a subset of original features. The original feature space of the data is then mapped onto a new feature space. *Feature selection in text mining* is addressed here separately due to the specificity of textual data compared to the data commonly addressed in machine learning.

Motivation and Background

Tasks addressed in machine learning on text are often characterized by a high number of features used to represent the data. However, these features are not necessarily all relevant and pbeneficial for the task and may slow down the applied methods giving similar results as a much smaller feature set. The main reasons for using feature selection in machine learning are Mladenić (2006) to improve performance, to improve learning efficiency, to provide faster models possibly requesting less information on the original data, and to reduce the complexity of the learned results and enable better understanding of the underlying process.

Feature selection in text mining was applied in a simple form from the start of applying machine learning methods on text data, for instance, feature selection by keeping the most frequent features and learning decision rules ▶ Rule Learning proposed in Apte et al. (1994) or keeping the most informative features for learning decision trees ▶ Decision Trees or ▶ naïve Bayes Bayes Rule proposed in Lewis and Ringuette (1994). The reason is that the number of features used to represent text data for machine learning tasks is high, as the basic approach of learning on text defines a feature for each word that occurs in the given text. This can easily result in several tens of thousands of features, compared to several tens or hundreds of features, as commonly observed on most machine learning tasks at the time.

Most methods for ▶ feature subset selection that are used on text are very simple compared to the feature selection methods developed in machine learning (Liu et al. 2010). They perform a filtering of features assuming feature independence, so that a score is assigned to each feature independently, and the features with high scores are selected. However, there are also more sophisticated methods for feature selection on text data that take into account interactions between the features. Embedded feature selection methods were successfully used on text data, either by applying a learning algorithm that has feature selection embedded (pre-processing step) or by inspecting a model generated by such an algorithm to extract feature scores. On the other hand, approaches to feature selection that search a space of all possible feature subsets can be rather time-consuming when dealing with a high number of features and are rarely used on text data.

Structure of Learning System

Feature selection in text mining is mainly used in connection with applying known machine learning and statistical methods on text when addressing tasks such as ▶ Document Clustering or ▶ Document Classification. This is also the focus of this chapter. However, we may need to perform some kind of feature selection on different text mining tasks where features are not necessary words or phrases, in which case we should reconsider the appropriate feature selection methods in the light of the task properties, including the number and type of features.

As already pointed out, the common way of document text representation is by defining a feature for each word in the document collection and feature selection by assuming feature independence, assigning score to the features, and selecting features with high scores. Scoring of individual features is performed either in an unsupervised way, ignoring the class information, or in a supervised way, taking into account the class information. Surprisingly both kind of approaches have been shown to perform comparably on ▶ document classification tasks, even though supervised scoring uses more information. Here we discuss several feature scoring measures and their performance on document classification, as reported in different researcher papers.

One of the first scoring measures used on text data is scoring by the number of documents that contain a particular word. This was applied after removing very frequent words, as given in a standard "stop-list" for English. An alternative is scoring by frequency – that is, by the number of times a feature occurs in a document collection. Both were shown to work well in document classification (Mladenić and Grobelnik 2003; Yang and Pedersen 1997).

▶ Information gain is commonly used in decision tree induction (Quinlan 1993). It was reported to work well as a feature scoring measure on text data (Yang and Pedersen 1997) in some domains (news articles of in a collection named Reuters-22173, abstracts of medical articles in a subset of the ▶ MEDLINE collection), where a multiclass problem was addressed using the nearest neighbor algorithm ▶ Nearest Neighbor. The same feature scoring almost completely failed when using ▶ naïve Bayes ▶ Bayes Rule on a binary classification problem on a hierarchical topic taxonomy of Web pages (Mladenić and Grobelnik 2003). This difference in performance can be partially attributed to the classification algorithm and domain characteristics.

It is interesting to notice that information gain takes into account all values for each feature. In the case of document classification, these are two values: occurs or does not occur in a document. On the other hand, expected cross entropy as used on text data (Koller and Sahami 1997; Mladenić and Grobelnik 2003) is similar in nature to information gain, but only uses the situation when the feature occurred in a document. Experiments on classifying document into a hierarchical topic taxonomy (Mladenić and Grobelnik 2003) have shown that this significantly improves performance. Expected cross entropy is related to information gain as follows: InfGain(F) = CrossEntropyTxt(F) + CrossEntropy Txt(F), where F is a binary feature (usually representing a word's occurrence).

The ▸ odds ratio was reported to outperform many other measures (Mladenić and Grobelnik 2003) in combination with naïve Bayes, used for document classification on data with highly imbalanced class distribution. A characteristic of naïve Bayes used for ▸ text classification is that, once the model has been generated, the classification is based on the features that occur in a document to be classified. This means that an empty document will be classified into the majority class. Consequently, having a highly imbalanced class distribution, if we want to identify documents from the underrepresented class value, we need to have a model sensitive to the features that occur in such documents. If most of the selected features are representative for the majority class value, the documents from other classes will be almost empty when represented using the selected features.

Experimental comparison of different feature selection measures in combination with the ▸ support vector machines ▸ Support Vector Machines classification algorithm ▸ (SVM) on news articles from the Reuters-2000 collection (Brank et al. 2002) has shown that using all or almost all the features yields the best performance. The same finding was confirmed in experimental evaluation of different feature selection measures on a number of text classification problems (Forman 2003). In addition, in Forman (2003) a new feature selection measure was introduced: binormal separation, which was reported to improve the performance of SVM, especially with problems where the class distribution is highly imbalanced. Interestingly, they also report that information gain is outperforming the other tested measures in the situation when using only a small number of selected features (20–50 features).

Another feature scoring measure for text data, called the Fisher index, was proposed as part of a ▸ document retrieval system based on organizing large text databases into hierarchical topic ▸ taxonomies (Chakrabarti et al. 1998). Similar to Mladenić (1998), for each internal node in the topic taxonomy, a separate feature subset is used to build a ▸ naïve Bayes model for that node. This is sometimes referred to as ▸ local feature selection or, alternatively, context-sensitive feature selection. The feature set used in each node is relatively small and tuned to the node context.

What follows are formulas of the described scoring measures as given in Mladenić and Grobelnik (2003).

$$\begin{aligned}\mathrm{InfGain}(F) &= P(F)\sum\nolimits_i P(C_i|F)\\ &\quad\times\log(P(C_i|F)/P(C_i)) + P(\bar{F})\sum\nolimits_i P(C_i|\bar{F})\\ &\quad\times\log P(C_i|\bar{F})/P(C_i))\end{aligned}$$

$$\mathrm{CrossEntropyTxt}(F) = P(F)\sum\nolimits_i P(C_i|F)\quad \log(P(C_i|F)/P(C_i))$$

$$\mathrm{MutualInfoTxt}(F) = \sum\nolimits_i P(C_i)\log(P(F|C_i)/P(F))$$

$$\begin{aligned}\mathrm{OddsRatio}(F) &= \log(P(F|C_{\mathrm{pos}})(1-P(F|C_{\mathrm{neg}})))\\ &\quad-\log((1-P(F|C_{\mathrm{pos}}))P(F|C_{\mathrm{neg}}))\end{aligned}$$

$$\begin{aligned}\mathrm{Bi\text{-}NormalSeparation}(F) &= Z^{-1}(P(F|C_{\mathrm{pos}}))\\ &\quad - Z^{-1}(P(F|C_{\mathrm{neg}}))\end{aligned}$$

$$\begin{aligned}\mathrm{FisherIndexTxt}(F) &= \left(\sum\nolimits_{\mathrm{pos,neg}}(P(F|C_{\mathrm{pos}}\right)\\ &\quad - P(F|C_{\mathrm{neg}}))^2)/\sum\nolimits_{\mathrm{Ci\varepsilon pos,neg}}|C_i|^{-1}\\ &\quad\times\sum\nolimits_{\mathrm{d\varepsilon Ci}}(n(F,d)-P(F|C_i))2\end{aligned}$$

where $P(F)$ is the probability that feature F occurred, $\bar{F}$ means that the feature does not occur, $P(C_i)$ is the probability of the ith class value, $P(C_i|F)$ is the conditional probability of the ith class value given that feature F occurred, $P(F|C_i)$ is the conditional probability of feature occurrence given the ith class value, $P(F|C_{\text{pos}})$ is the conditional probability of feature F occurring given the class value "positive," $P(F|C_{\text{neg}})$ is the conditional probability of feature F occurring given the class value "negative," $Z^{-1}(x)$ is the standard normal distribution's inverse ▶ cumulative probability function (z-score), $|C_i|$ is the number of documents in class C_i , and $n(F, d)$ is 1 if the document d contains feature F and 0 otherwise.

As already highlighted in ▶ text classification, most of the feature selection methods evaluate each feature independently. A more sophisticated approach is proposed in Brank et al. (2002), where a linear SVM is first trained using all the features, and the induced model is then used to score the features (weight assigned to each feature in the normal to the induced hyperplane is used as a feature score). Experimental evaluation using that feature selection in combination with ▶ SVM, Perceptron, and naïve Bayes has shown that the best performance is achieved by SVM when using almost all the features. The experiments have confirmed the previous findings on ▶ feature subset selection improving the performance of naïve Bayes, but the overall performance is lower than using SVM on all the features.

Much the same as in Brank et al. (2002), feature selection was performed using a linear SVM to rank the features in Bi et al. (2003). However, the experiments in Bi et al. (2003) were performed on a regression problem, and the final model was induced using a nonlinear SVM. The feature selection was shown to improve performance. Feature selection on text was also performed in a two-stage way, first using information gain to score the features and then applying genetic algorithms and principal component analysis (Uguz 2011) .

Distributional clustering of words with an agglomerative approach (words are viewed as distributions over document categories) is used for dimensionality reduction via ▶ feature construction (Bekkerman et al. 2003) that preserves the mutual information between the features as much as possible. This representation was shown to achieve comparable or better results than the bag-of-words document representation using feature selection based on mutual information for text; a linear SVM was used as the classifier. A related approach, also based on preserving the mutual information between the features (Globerson and Tishby 2003), finds new dimensions by using an iterative projection algorithm instead of clustering. It was shown to achieve performance comparable to the bag-of-words representation with all the original features, using significantly less features (e.g., on one dataset, four constructed features achieved 98 % of performance of 500 original features) using the linear SVM classifier.

Divisive clustering for feature construction (Dhillon et al. 2003) was shown to outperform distributional clustering when used for dimensionality reduction on text data. The approach uses the Kullback-Leibler divergence as a distance function and minimizes within-cluster divergence while maximizing between-cluster divergence. Experiments on two datasets have shown that this dimensionality reduction slightly improves the performance of ▶ naïve Bayes (compared to using all the original features), outperforming the agglomerative clustering of words combined with naïve Bayes and achieving considerably higher classification accuracy for the same number of features than feature subset selection using information gain or mutual information (in combination with naïve Bayes or SVM).

Recommended Reading

Apte C, Damerau F, Weiss SM (1994) Toward language independent automated learning of text categorization models. In: Proceedings of the 17th annual international ACM SIGIR conference on research and development in information retrieval, Dublin, pp 23–30

Bekkerman R, El-Yaniv R, Tishby N, Winter Y (2003) Distributional word clusters vs. words for text categorization. J Mach Learn Res 3:1183–1208

Bi J, Bennett KP, Embrechts M, Breneman CM, Song M (2003) Dimensionality reduction via sparse support vector machines. J Mach Learn Res 3: 1229–1243
Brank J, Grobelnik M, Milič-Frayling N, Mladenić D (2002) Feature selection using support vector machines. In: Zanasi A (ed) Data mining III, Southampton, pp 261–273
Chakrabarti S, Dom B, Agrawal R, Raghavan P (1998) Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. VLDB J 7: 163–178
Dhillon I, Mallela S, Kumar R (2003) A divisive information-theoretic feature clustering algorithm for text classification. J Mach Learn Res 3: 1265–1287
Forman G (2003) An extensive empirical study of feature selection metrics for text classification. J Mach Learn Res 3:1289–1305
Globerson A, Tishby N (2003) Sufficient dimensionality reduction. J Mach Learn Res 3:1307–1331
Koller D, Sahami M (1997) Hierarchically classifying documents using very few words. In: Proceedings of the 14th international conference on machine learning ICML'97, Nashrille, pp 170–178
Lewis DD, Ringuette M (1994) Comparison of two learning algorithms for text categorization. In: Proceedings of the 3rd annual symposium on document analysis and information retrieval SDAIR-1994, Las Vegas
Liu H, Motodo H, Setiono R, Zhao Z (2010) Feature selection: an ever evolving frontier in data mining. In: Proceedings of the fourth workshop on feature selection in data mining, pp 4–13
Mladenić D (1998) Feature subset selection in text-learning. In: Proceedings of the 10th European conference on machine learning ECML'98, Chemnitz
Mladenić D (2006) Feature selection for dimensionality reduction. In: Saunders C, Gunn S, Shawe-Taylor J, Grobelink M (eds) Subspace, latent structure and feature selection: statistical and optimization perspectives workshop. Lecture notes in computer science, vol 3940. Springer, Berlin/Heidelberg, pp 84–102
Mladenić D, Grobelnik M (2003) Feature selection on hierarchy of web documents. J Decis Support Syst 35:45–87
Quinlan JR (1993) Constructing decision tree. In: Quinlan JR (ed) C4.5: programs for machine learning. Morgan Kaufmann Publishers, San Francisco
Uguz H (2011) A two-stage feature selection method for text categorization by using information gain, principal component analysis and genetic algorithm. Knowl-Based Syst 24(7):1024–1032
Yang Y, Pedersen JO (1997) A comparative study on feature selection in text categorization. In: Proceedings of the 14th international conference on machine learning ICML'97, Las Vegas, pp 412–420

Feature Subset Selection

▶ Feature Selection

Feature Weighting

▶ Feature Selection

Feedforward Recurrent Network

▶ Simple Recurrent Network

Field Scrubbing

▶ Record Linkage

Finite Mixture Model

▶ Mixture Model

First-Order Logic

Peter A. Flach
Department of Computer Science, University of Bristol, Bristol, UK

Synonyms

Predicate logic; Predicate calculus; First-order predicate logic; First-order predicate calculus

Definition

First-order predicate logic – first-order logic for short – is the logic of properties of, and relations between, objects and their parts. Like any logic, it consists of three parts: *syntax* governs the formation of *well-formed formulae*, *semantics* ascribes meaning to well-formed formulae and formalizes the notion of *deductive consequence*, and *proof procedures* allow the inference of deductive consequences by syntactic means. A number of variants of first-order logic exist, mainly differing in their syntax and proof systems. In machine learning, the main use of first-order logic is in ▸ Learning from Structured Data, ▸ Inductive Logic Programming, and ▸ Relational Data Mining.

Motivation and Background

The interest in logic arises from a desire to formalize human, mathematical, and scientific reasoning and goes back to at least the Greek philosophers. Aristotle devised a form of propositional reasoning called *syllogisms* in the fourth century BC. Aristotle was held in very high esteem by medieval scholars, and so further significant advances were not made until after the Middle Ages. Leibniz wrote of an "algebra of thought" and linked reasoning to calculation in the late seventeenth century. Boole and De Morgan developed the algebraic point of view in the mid-nineteenth century.

Universally quantified variables, which form the main innovation in first-order logic as compared to ▸ Propositional Logic, were invented by Gottlob Frege in his *Begriffsschrift* ("concept notation") from 1879 and independently by Charles Sanders Peirce in 1885, who introduced the notation $\prod_x$ and $\sum_x$ for universal and existential quantification. Frege's work went largely unnoticed until it was developed further by Alfred North Whitehead and Bertrand Russell in their *Principia Mathematica* (1903). Seminal contributions were made, among many others: by Giuseppe Peano, who axiomatized number theory and introduced the notation (x) and $\exists x$; by Kurt Gödel, who established the completeness of first-order logic as well as the incompleteness of any system incorporating Peano arithmetic; by Alonzo Church, who proved that first-order logic is undecidable and who introduced λ-calculus, a form of ▸ Higher-Order Logic that allows quantification over predicates and functions (as opposed to first-order logic, which only allows quantification over objects); and by Alfred Tarski, who pioneered logical semantics through model theory and the notion of logical consequence. The now universally accepted notation $\forall x$ was introduced by Gerhard Gentzen.

Logic plays an important role in any approach to symbolic AI that employs a formal language for knowledge representation and inference. A significant, relatively recent development was the introduction of logic programming languages such as ▸ Prolog, which turn logical inference into computation. In machine learning, the use of a first-order language is essential in order to handle domains in which objects have inherent structure; the availability of Prolog as a common language and programming platform gave rise to the field of ▸ Inductive Logic Programming.

Theory

Syntax

A first-order logical language is built from *constant symbols*, *variable symbols*, *predicate symbols*, and *function symbols*; the latter two kinds of symbols have an associated *arity*, which is the number of arguments they take. *Terms* are either constant symbols, variable symbols, or of the form $f(t_1, \ldots, t_n)$ where f is a function symbol with arity n and $t_1, \ldots, t_n$ is a sequence of n terms. Using the logical connectives $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), and $\rightarrow$ (material implication) and the quantifiers $\forall$ (universal quantifier) and $\exists$ (existential quantifier), well-formed formulae or *wffs* are defined recursively as follows: (1) if P is a predicate symbol with arity n, and $t_1, \ldots, t_n$ is a sequence of n terms, then $P(t_1, \ldots, t_n)$ is a wff, also referred to as an atomic formula or *atom*; (2) if ϕ_1 and ϕ_2 are wffs, then $(\neg\phi_1)$, $(\phi_1 \wedge \phi_2)$, $(\phi_1 \vee \phi_2)$, and $(\phi_1 \rightarrow \phi_2)$ are wffs; (3) if x is a variable and ϕ is a wff, then $(\forall x : \phi)$ and $(\exists x : \phi)$ are wffs; (4) and nothing else is a wff. Brackets are usually dropped as much as it is possible without causing confusion.

Example 1 Let "*man*," "*single*," and "*partner*" be two unary and one binary predicate symbol, respectively, and let "x" and "y" be variable symbols, then the following is a wff ϕ expressing that men who are not single have a partner:

$$(\forall x : (man(x) \wedge (\neg single(x))) \\ \rightarrow (\exists y : partner(x, y)))$$

Assuming that $\neg$ binds strongest, then $\wedge$, then $\rightarrow$, the brackets can be dropped:

$$\forall x : man(x) \wedge \neg single(x) \\ \rightarrow \exists y : partner(x, y)$$

A *propositional language* is a special case of a predicate-logical language, built only from predicate symbols with arity 0, referred to as *proposition symbols* or propositional atoms, and connectives. So, for instance, assuming the proposition symbols "*man*," "*single*," and "*has_partner*," the following is a propositional wff: $man \wedge \neg single \rightarrow has\_partner$. The main difference is that in propositional logic, references to objects cannot be expressed and therefore have to be understood implicitly.

Semantics

First-order wffs express statements that can be true or false, and so a first-order semantics consists in constructing a mapping from wffs to truth values, given an interpretation, which is a possible state of affairs in the domain of discourse, mapping constant, predicate, and function symbols to elements, relations, and functions in and over the domain. To deal with variables, a valuation function is employed. Once this mapping is defined, the meaning of a wff consists in the set of interpretations in which the wff maps to true, also called its models. The intuition is that the more "knowledge" a wff contains, the fewer models it has. The key notion of logical consequence is then defined in terms of models: one wff is a logical consequence of another if the set of models of the first contains the set of models of the second; hence, the second wff contains at least the same, if not more, knowledge than the first.

Formally, a *predicate-logical interpretation*, or *interpretation* for short, is a pair (D, i), where D is a nonempty *domain* of individuals and i is a function assigning to every constant symbol an element of D, to every function symbol with arity n a mapping from D^n to D, and to every predicate symbol with arity n a subset of D^n, called the *extension* of the predicate. A *valuation* is a function v assigning to every variable symbol an element of D.

Given an interpretation $I = (D, i)$ and a valuation v, a mapping i_v from terms to individuals is defined as follows: (1) if t is a constant symbol, $i_v(t) = i(t)$; (2) if t is a variable symbol, $i_v(t) = v(t)$; (3) and if t is a term $f(t_1, \ldots, t_n)$, $i_v(t) = i(f)(i_v(t_1), \ldots, i_v(t_n))$. The mapping is extended to a mapping from wffs to truth values as follows: (4) if ϕ is an atom $P(t_1, \ldots, t_n)$, $i_v(\phi) = i(P)(i_v(t_1), \ldots, i_v(t_n))$; (5) $i_v(\neg\phi) = T$ if $i_v(\phi) = F$ and F otherwise; (6) $i_v(\phi_1 \wedge \phi_2) = T$ if $i_v(\phi_1) = T$ and $i_v(\phi_2) = T$ and F otherwise; (7) and $i_v(\forall x : \phi) = T$ if $i_{v_{x \rightarrow d}}(\phi) = T$ for all $d \in D$ and F otherwise, where $v_{x \rightarrow d}$ is v except that x is assigned d. The remaining connectives and quantifier are evaluated by rewriting: (8) $i_v(\phi_1 \vee \phi_2) = i_v(\neg(\neg\phi_1 \wedge \neg\phi_2))$; (9) $i_v(\phi_1 \rightarrow \phi_2) = i_v(\neg\phi_1 \vee \phi_2)$; (10) $i_v(\exists x : \phi) = i_v(\neg\forall x : \neg\phi)$.

An interpretation I *satisfies* a wff ϕ, notation $I \models \phi$, if $i_v(\phi) = T$ for all valuations v; we say that I is a *model* of ϕ and that ϕ is *satisfiable*. If all models of a set of wffs Σ are also models of ϕ, we say that Σ *logically entails* ϕ or ϕ is a *logical consequence* of Σ and write $\Sigma \models \phi$. If $\Sigma = \emptyset$, ϕ is called a *tautology* and we write $\models \phi$. A wff ψ is a *contradiction* if $\neg\psi$ is a tautology. Contradictions do not have any models, and consequently $\psi \models \alpha$ for any wff α. The *deduction theorem* says that $\Sigma \models \alpha \rightarrow \beta$ if and only if $\Sigma \cup \{\alpha\} \models \beta$. Another useful fact is that, if $\Sigma \cup \{\neg\gamma\}$ is a contradiction, $\Sigma \models \gamma$; this gives rise to a proof technique known as *Reductio ad absurdum* or *proof by contradiction* (see below).

Example 2 We continue the previous example. Let $D = \{Peter, Paul, Mary\}$, and let the function i be defined as follows: $i(man) = \{Peter, Paul\}$; $i(single) = \{Paul\}$; $i(partner) = \{(Peter, Mary)\}$. We then have that the interpretation $I = (D, i)$ is a model for the wff ϕ above. On the other hand, I does not satisfy $\psi = \forall x : \exists y : partner(x, y)$, and therefore $\phi \not\models \psi$. However, the reverse does hold: there is no interpretation that satisfies ψ and not ϕ, and therefore $\psi \models \phi$.

In case of a propositional logic, this semantics can be considerably simplified. Since there are no terms, the domain D plays no role, and an interpretation simply assigns truth values to proposition symbols. Wffs can then be evaluated using rules (5–6) and (8–9). For example, if $i(man) = T$, $i(single) = T$, and $i(has\_partner) = T$, then $i(man \wedge \neg single \rightarrow has\_partner) = T$. (If this seems counterintuitive, this is probably because the reader's knowledge of the domain suggests another wff $\neg(single \wedge has\_partner)$, which is false in this particular interpretation.)

Proofs

A *proof procedure* consists of a set of *axioms* and a set of *inference rules*. Given a proof procedure P, we say that ϕ is *provable* from Σ and write $\Sigma \vdash_P \phi$ if there exists a finite sequence of wffs $\phi_1, \phi_2, \ldots, \phi_{n-1}, \phi$ which is obtained by successive applications of inference rules to axioms, *premisses* in Σ, and/or previous wffs in the sequence. Such a sequence of wffs, if it exists, is called a *proof* of ϕ from Σ. A proof procedure P is *sound*, with respect to the semantics established by predicate-logical interpretations, if $\Sigma \models \phi$ whenever $\Sigma \vdash_P \phi$; it is *complete* if $\Sigma \vdash_P \phi$ whenever $\Sigma \models \phi$. For a sound and complete proof procedure for first-order predicate logic, see, e.g., Turner (1984, p.15).

A set of wffs Σ is *consistent*, with respect to a proof procedure P, if not both $\Sigma \vdash_P \phi$ and $\Sigma \vdash_P \neg\phi$ for some wff ϕ. Given a sound and complete proof procedure, the proof-theoretic notion of consistency coincides with the semantic notion of satisfiability. In particular, if we can prove that $\Sigma \cup \{\neg\gamma\}$ is inconsistent, then we know that $\Sigma \cup \{\neg\gamma\}$ is not satisfiable, hence a contradiction, and thus $\Sigma \models \gamma$. This still holds if the proof procedure is only complete in the weaker sense of being able to demonstrate the inconsistency of arbitrary sets of wffs (see the resolution inference rule, below).

Example 3 One useful inference rule for predicate logic replaces a universally quantified variable with an arbitrary term, which is called *Universal Elimination*. So, if "c" is a constant symbol in our language, then we can infer

$$man(c) \wedge \neg single(c) \rightarrow \exists y : partner(c, y)$$

from ϕ above by Universal Elimination. Another inference rule, which was called *Modus Ponens* by Aristotle, allows us to infer β from α and $\alpha \rightarrow \beta$. So, if we additionally have $man(c) \wedge \neg single(c)$, then we can conclude

$$\exists y : partner(c, y)$$

by Modus Ponens. This rule is also applicable to propositional logic. An example of an axiom is $c = c$ for any constant symbol c. (Strictly speaking this is an *axiom schema*, giving rise to an axiom for every constant symbol in the language.)

Programming in Logic

Syntax, semantics, and proof procedures for first-order logic can be simplified and made more amenable to computation if we limit the number of ways of expressing the same thing. This can be achieved by restricting wffs to a normal form called *prenex conjunctive normal form* (PCNF). This means that all quantifiers occur at the start of the wff and are followed by a conjunction of disjunctions of atoms and negated atoms, jointly called *literals*. An example of a formula in PCNF is

$$\forall x : \exists y : \neg man(x) \vee single(x) \vee partner(x, y)$$

This formula is equivalent to the wff ϕ in Example 1, in the sense that it has the same set of models, and so either one logically entails the other.

Every first-order wff can be transformed into a logically equivalent formula in PCNF, which is unique up to the order of conjuncts and disjuncts. A transformation procedure can be found in Flach (1994).

PCNF can be further simplified if we use function symbols instead of existential quantifiers. For instance, instead of $\exists y : partner(x, y)$, we can say $partner(x, partner\_of(x))$, where *partner_of* is a unary function symbol called a *Skolem function*, after the Norwegian logician Thoralf Skolem. The two statements are not logically equivalent, as the second entails the first but not vice versa, but this difference is of little practical consequence. Since all variables are now universally quantified, the quantifiers are usually omitted, leading to *clausal form*:

$$\neg man(x) \vee single(x) \\ \vee\ partner(x, partner\_of(x))$$

To sum up, a wff in clausal form is a conjunction of disjunctions of literals, of which the variables are implicitly universally quantified. The individual disjunctions are called *clauses*.

Further simplifications include dispensing with equality, which means that terms involving function symbols, such as $partner\_of(c)$, are not evaluated and in effect treated as names of objects (in this case, the function symbols are called *functors* or *data constructors*). Under this assumption each *ground term* (a term without variables) denotes a different object, which means that we can take the set of ground terms as the domain D of an interpretation; this is called a *Herbrand interpretation*, after the French logician Jacques Herbrand.

The main advantage of clausal logic is the existence of a proof procedure consisting of a single inference rule and no axioms. This inference rule, which is called *resolution*, was introduced by Alan Robinson in 1965 (Robinson 1965). In propositional logic, given two clauses $P \vee Q$ and $\neg Q \vee R$ containing *complementary* literals Q and $\neg Q$, resolution infers the *resolvent* $P \vee R$ (P and/or R may themselves contain several disjuncts). For instance, given $\neg man \vee single \vee has\_partner$ and $man \vee woman$, we can infer $woman \vee single \vee has\_partner$ by resolution. In first-order logic, Q and $\neg Q'$ are complementary if Q and Q' are *unifiable*, i.e., there exists a *substitution* θ of terms for variables such that $Q\theta = Q'\theta$, where $Q\theta$ denotes the application of substitution θ to Q; in this case, the resolvent of $P \vee Q$ and $\neg Q' \vee R$ is $P\theta \vee R\theta$. For instance, from the following two clauses:

$$\neg man(x) \vee single(x) \\ \vee\ partner(x, partner\_of(x))$$

$$\neg single(father\_of(c))$$

we can infer

$$\neg man(father\_of(c)) \vee partner(father\_of(c), \\ partner\_of(father\_of(c)))$$

The resolution inference rule is sound but not complete: for instance, it is unable to produce tautologies such as $man(c) \vee \neg man(c)$ if no clauses involving the predicate man are given. However, it is *refutation-complete*, which means it can demonstrate the unsatisfiability of any set of clauses by deriving the *empty clause*, indicated by $\square$. For instance, $man(c) \wedge \neg man(c)$ is a wff consisting of two clauses which are complementary literals, so by resolution we infer the empty clause in one step.

Refutation by resolution is the way in which queries are answered in the logic programming language ▶ Prolog. Prolog works with a subset of clausal logic called *Horn logic*, named after the logician Alfred Horn. A *Horn clause* is a disjunction of literals with at most one positive (un-negated) literal; Horn clauses can be further divided into *definite clauses*, which have one positive literal, and *goal clauses* which have none. A Prolog program consists of definite clauses, and a goal clause functions as a procedure call. Notice that resolving a goal clause with a definite clause results in another goal clause, because the positive literal in the definite clause (also called its *head*) must be one of the complementary literals. The idea is that the resolution step reformulates the original goal into a new goal that is one step closer to the solution. A refutation is then a sequence of goals $G, G_1, G_2, \ldots, G_n$ such that G is the original goal, each G_i is obtained by resolving G_{i-1} with a clause from the program P, and $G_n = \Box$. Such a refutation demonstrates that $P \cup \{G\}$ is inconsistent, and therefore $P \models \neg G$.

Finding a refutation amounts to a search problem, because there are typically several program clauses that could be resolved against the current goal. Virtually all Prolog interpreters apply a depth-first search procedure, searching the goal literals left to right and the program clauses top-down. Once a refutation is found, the substitutions collected in all resolution steps are composed to obtain an *answer substitution*. One unique feature of logic programming is that a goal may have more than one (or, indeed, less than one) refutation and answer substitution from a given program.

Example 4 Consider the following Prolog program:

```
peano_sum(0,Y,Y).
peano_sum(s(X),Y,s(Z)):
  -peano_sum(X,Y,Z).
```

This program defines addition in Peano arithmetic. We follow Prolog syntax: variables start with an uppercase letter, and `:-` stands for reversed implication ← or "if." The unary functor `s` represents the successor function. So the first rule reads "the sum of 0 and an arbitrary number y is y," and the second rule reads "the sum of $x + 1$ and y is $z + 1$ if the sum of x and y is z."

The goal `:-peano_sum(s(0),s(s(0)),Q)` states, "there are no numbers q such that $1 + 2 = q$." We first resolve this goal with the second program clause to obtain `:-peano_sum(0,s(s(0)),Z)` under the substitution {`Q / s(Z)`}. This new goal states, "there are no numbers z such that $0 + 2 = z$." It is resolved with the first clause to yield the empty clause under the substitution {`Y / s(s(0)),Z / s(s(0))`}. The resulting answer substitution is {`Q / s(s(s(0)))`}, i.e., $q = 3$.

As another example, goal `:-peano_sum(A,B,s(s(0)))` states "there are no numbers a and b such that $a + b = 2$." This goal has three refutations: one involving the first clause only, yielding the answer substitution {`A / 0, B / s(s(0))`}; one involving the second clause then the first, resulting in {`A / s(0), B / s(0)`}; and the third applying the second clause twice followed by the first, yielding {`A / s(s(0)), B / 0`}. Prolog will return these three answers in this order.

Induction in first-order logic amount to reconstructing a logical theory from some of its logical consequences. For techniques to induce a Prolog program given examples such as `peano_sum(s(0),s(0),s(s(0)))`, see ▸ Inductive Logic Programming.

Cross-References

▸ Abduction
▸ Entailment
▸ Higher-Order Logic
▸ Hypothesis Language
▸ Inductive Logic Programming
▸ Learning from Structured Data
▸ Propositionalization
▸ Relational Data Mining

Recommended Reading

For general introductions to logic and its use in Artificial Intelligence, see Turner (1984) and Genesereth and Nilsson (1987). Kowalski's classic text *Logic for problem solving* focusses on clausal logic and resolution theorem proving (Kowalski 1979). For introductions to Prolog programming, see Flach (1994) and Bratko (2001).

Bratko I (2001) Prolog programming for artificial intelligence, 3rd edn. Addison Wesley, Harlow/New York

Flach P (1994) Simply logical: intelligent reasoning by example. Wiley, Chichester/New York

Genesereth MR, Nilsson NJ (1987) Logical foundations of artificial intelligence. Morgan Kaufmann, Los Altos

Kowalski RA (1979) Logic for problem solving. North-Holland, New York

Robinson JA (1965) A machine-oriented logic based on the resolution principle. J ACM 12(1):23–41

Turner R (1984) Logics for artificial intelligence. Ellis Horwood, Chichester

First-Order Predicate Calculus

▶ First-Order Logic

First-Order Predicate Logic

▶ First-Order Logic

First-Order Regression Tree

Synonyms

Logical regression tree; Relational regression tree

Definition

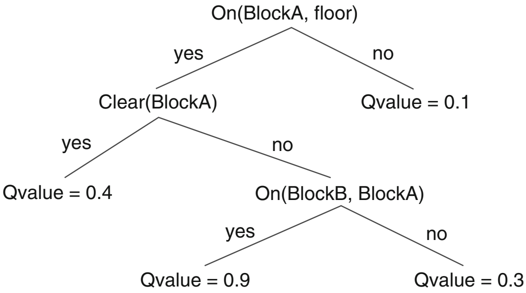
A first-order regression tree can be defined as follows:

Definition 1 (First-Order Regression Tree) A first-order regression tree is a binary tree in which

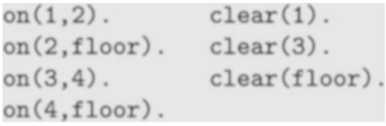
- Every internal node contains a test which is a conjunction of first-order literals.
- Every leaf (terminal node) of the tree contains a real valued prediction.

An extra constraint placed on the first-order literals that are used as tests in internal nodes is that a variable that is introduced in a node (i.e., it does not occur in higher nodes) does not occur in the right subtree of the node.

Figure 1 gives an example of a first-order regression tree. The test in a node should be read as the existentially quantified conjunction of all literals in the nodes in the path from the root of the tree to that node. In the left subtree of a node, the test of the node is added to the conjunction, for the right subtree, the negation of the test should be added. For the example state description of Fig. 2, the tree would predict a $Qvalue = 0.9$, since there exists no block that is both on the floor and clear, but there is a block which is on the floor and has another block on top of it. To see this, substitute BlockA in the tree with 2 (or 4) and BlockB with 1 (or 4).



First-Order Regression Tree, Fig. 1 A relational regression tree



```
on(1,2).          clear(1).
on(2,floor).      clear(3).
on(3,4).          clear(floor).
on(4,floor).
```

First-Order Regression Tree, Fig. 2 State description

The constraint on the use of variables stems from the fact that variables in the tests of internal nodes are existentially quantified. Suppose a node introduces a new variable X. Where the left subtree of a node corresponds to the fact that a substitution for X has been found to make the conjunction true, the right side corresponds to the situation where no substitution for X exists, i.e., there is no such X. Therefore, it makes no sense to refer to X in the right subtree.

Cross-References

▶ First-Order Logic
▶ Inductive Logic Programming
▶ Relational Reinforcement Learning

Formal Concept Analysis

Gemma C. Garriga
Universite Pierre et Marie Curie, Paris, France

Definition

Formal concept analysis is a mathematical theory of concept hierarchies that builds on order theory; it can be seen as an unsupervised machine learning technique and is typically used as a method of knowledge representation. The approach takes an input binary relation (binary matrix) specifying a set of objects (rows) and a set of attributes for those objects (columns), finds the natural concepts described in the data, and then organizes the concepts in a partial order structure or Hasse diagram. Each concept in the final diagram is a pair of sets of objects and attributes that are maximally contained one in each other.

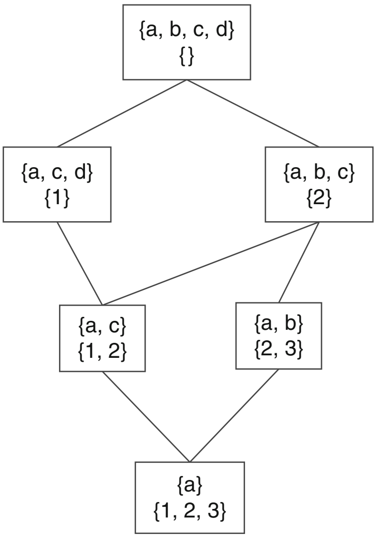
Theory

The above intuition can be formalized through a Galois connection as follows. Let R be the binary relation between a set of objects and a set of attributes, that is, $R \subseteq \mathcal{O} \times \mathcal{A}$. Two mappings $\alpha\colon \mathcal{O} \mapsto \mathcal{A}$ and $\beta\colon \mathcal{A} \mapsto \mathcal{O}$ are defined so that the operator $\alpha(O)$, for some $O \subseteq \mathcal{O}$, returns the maximal set of attributes common to all objects in O; dually, the operator $\beta(A)$, for some $A \subseteq \mathcal{A}$, returns the maximal set of objects containing all attributes in A. there two mappings induce a Galois connection between the powerset of objects and the powerset of attributes, that is, they satisfy $O \subseteq \beta(A) \Leftrightarrow A \subseteq \alpha(0)$ for a set of objects O and a set of attributes A.

From here, a formal concept is a pair of sets of objects and attributes (O, A) from the binary relation that satisfy $\alpha(O) = A$ and $\beta(A) = O$. Typically, O is called the extent of the concept and A the intent of the concept. Note that concepts can be interpreted from the geometrical point of view, they are maximal rectangles of ones (not necessarily consecutive) in the input binary table R. The organization of all the formal concepts in a Hasse diagram is called the concept lattice. This lattice corresponds to a partial order structure of concepts where edges between concepts correspond to the standard inclusion of the sets.

	a	b	c	d
1	1	0	1	1
2	1	1	1	0
3	1	1	0	0

Formal Concept Analysis, Fig. 1 A binary relation $R \subseteq \{1, 2, 3\} \times \{a, b, c, d\}$



Formal Concept Analysis, Fig. 2 Concepts of the relation R organized in a Hasse diagram

A small toy example in Figs. 1 and 2 illustrates the formal concepts and their organization in a Hasse diagram.

Motivation and Background

Formal concept analysis has been applied to a variety of disciplines, from psychology, sociology, biology, medicine, linguistics, or industrial engineering, to cite some, for the interactive exploration of implicit and explicit structures in the data.

From the point of view of machine learning and data mining, the connection between the formal concepts of the lattice and the so-called, closed sets of items is remarkable. Closed sets of items appear in the context of ▸ constraint-based mining, in which the user provides restraints that guide a search of patterns in the data. They are maximal sets of attributes occuring frequently in the data; they correspond to a compacted representation of the frequent sets from frequent itemset mining. It is well known that closed sets correspond exactly to the intents of the concepts derived via formal concept analysis, and therefore, from the formal concepts it is possible to construct bases of minimal nonredundant sets of association rules from which all other rules holding in the data can be derived.

Also, formal concept analysis has been typically seen as a type of conceptual ▸ clustering. Each concept or groups of concepts form a cluster of objects sharing similar properties. The diagrams obtained from this sort of clustering can then be used in class discovery and class prediction. Although a diagram of concepts can become large and complex, different approaches have worked toward reducing the complexity of concept lattices via conceptual scaling.

We refer the reader to Ganter and Wille (1998) for a general reference on formal concept analysis, and to Davey and Priestly (2002) for the basic concepts on order theory. For more thorough descriptions of different applications of formal concept analysis in the computer science field, see Carpineto and Romano (2004).

Cross-References

▸ Clustering
▸ Constraint-Based Mining
▸ Frequent Itemset

Recommended Reading

Carpineto C, Romano G (2004) Concept data analysis. Theory and applications. Wiley, New York

Davey BA, Priestly HA (2002) Introduction to lattices and order. Cambridge University Press, Cambridge

Ganter B, Wille R (1998) Formal concept analysis. Mathematical foundations. Springer, Heidelberg

Frequent Itemset

Hannu Toivonen
University of Helsinki, Helsinki, Finland

Synonyms

Frequent set

Definition

Frequent itemsets (Agrawal et al. 1993, 1996) are a form of ▸ frequent pattern. Given examples that are sets of items and a minimum frequency, any set of items that occurs at least in the minimum number of examples is a frequent itemset.

For instance, customers of an on-line bookstore could be considered examples, each represented by the set of books he or she has purchased. A set of books, such as {"*Machine Learning*," "*The Elements of Statistical Learning*," "*Pattern Classification*,"} is a frequent itemset if it has been bought by sufficiently many customers. Given a frequency threshold, perhaps only 0.1 or 0.01 % for an on-line store, *all* sets of books that have been bought by at least that many customers are called frequent. Discovery of all frequent itemsets is a typical data mining task. The original use has been as part of ▸ association rule discovery. ▸ Apriori is a classical algorithm for finding frequent itemsets.

The idea generalizes far beyond examples consisting of sets. The pattern class can be re-defined, e.g., to be (frequent) subsequences rather than itemsets; or original data can often be transformed to a suitable representation, e.g., by considering each discrete attribute-value pair or an interval of a continuous attribute as an individual item. In such more general settings, the term ▸ frequent pattern is often used. Another direction to generalize frequent itemsets is to consider other conditions than frequency on the patterns to be discovered; see ▸ constraint-based mining for more details.

Cross-References

- ▶ Apriori Algorithm
- ▶ Association Rule
- ▶ Constraint-Based Mining
- ▶ Frequent Pattern

Recommended Reading

Agrawal R, Imieliski T, Swami A (1993) Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD international conference on management of data, Washington, DC. ACM, New York, pp 207–216

Agrawal R, Mannila H, Srikant R, Toivonen H, Verkamo AI (1996) Fast discovery of association rules. In: Fayyad UM, Piatetsky-Shapiro G, Smyth P, Uthurusamy R (eds) Advances in knowledge discovery and data mining. AAAI Press, Menlo Park, pp 307–328

Frequent Pattern

Hannu Toivonen
University of Helsinki, Helsinki, Finland

Definition

Given a set $\mathcal{D}$ of examples, a language $\mathcal{L}$ possible patterns, and a minimum frequency min_{-fr}, every pattern $\theta \in \mathcal{L}$ that occurs at least in the minimum number of examples, i.e., $|\{ e \in \mathcal{D} | \theta$ occurs in $e\}| \geq min_{-fr}$, is a frequent pattern. Discovery of all frequent patterns is a common data mining task. In its most typical form, the patterns are ▶ frequent itemsets. A more general formulation of the problem is ▶ constraint-based mining.

Motivation and Background

Frequent patterns can be used to characterize a given set of examples: they are the most typical feature combinations in the data.

Frequent patterns are often used as components in larger data mining or machine learning tasks. In particular, discovery of ▶ frequent itemsets was actually first introduced as an intermediate step in ▶ association rule mining (Agrawal et al. 1993) ("frequent itemsets" were then called "large"). The frequency and confidence of every valid association rule $X \rightarrow Y$ are obtained simply as the frequency of $X \cup Y$ and the ratio of frequencies of $X \cup Y$ and X, respectively.

Frequent patterns can be useful as ▶ features for further learning tasks. They may capture shared properties of examples better than individual original features, while the frequency threshold gives some guarantee that the constructed features are not so likely just noise. However, other criteria besides frequency are often used to choose a good set of candidate patterns.

Structure of Problem

A frequent pattern often is essentially a set of binary ▶ features. Given a set $\mathcal{I}$ of all available features, the pattern language $\mathcal{L}$ then is the power set of $\mathcal{I}$. An example in data $\mathcal{D}$ covers a pattern $\theta \in \mathcal{L}$ if it has all the features of θ. In such cases, the frequent pattern discovery task reduces to the task of discovering ▶ frequent itemsets. Therefore, the structure of the frequent pattern discovery problem is best described using the elementary case of frequent itemsets.

Let $\mathcal{I}$ be the set of all items (or binary features); sub-sets of $\mathcal{I}$ are called itemsets (or examples or patterns, depending on the context). The input to the frequent itemset mining problem is a multiset $\mathcal{D}$ of itemsets (examples described by their features), and a frequency threshold. The task is to output *all* frequent itemsets (patterns) and their frequencies, i.e., all subsets of $\mathcal{I}$ that exceed the given frequency threshold in the given data $\mathcal{D}$.

Example 1 Assume the following problem specification:

- Set of all items $\mathcal{I} = \{A, B, C, D\}$.
- Data $\mathcal{D} = \{\{A, B, C\}\{A, D\}, \{B, C, D\}, \{A, B, C\}, \{C, D\}, \{B, C\}\}$.
- Frequency threshold is 2.

All possible itemsets and their frequencies:

Itemset	Frequency
$\{A\}$	**3**
$\{B\}$	**4**
$\{C\}$	**5**
$\{D\}$	**3**
$\{A, B\}$	**2**
$\{A, C\}$	**2**
$\{A, D\}$	1
$\{B, C\}$	**4**

Itemset	Frequency
$\{B, D\}$	1
$\{C, D\}$	**2**
$\{A, B, C\}$	**2**
$\{A, B, D\}$	0
$\{A, C, D\}$	0
$\{B, C, D\}$	1
$\{A, B, C, D\}$	0

The frequent itemsets are $\{A\}, \{B\}, \{C\}, \{D\}$, $\{A, B\}, \{A, C\}, \{B, C\}, \{C, D\}, \{A, B, C\}$.

The ▶ hypothesis space for itemsets obviously is the power set of $\mathcal{I}$, and it has an exponential size ($2^{|\mathcal{I}|}$) in the number of items. Since all frequent itemsets are output, this is also the size of the output in the worst case (e.g., if the frequency threshold is zero, or if all examples in $\mathcal{D}$ equal $\mathcal{I}$), as well as the worst case time complexity.

In practical applications of frequent itemset mining, the size of the output as well as the running times are much smaller, but they strongly depend on the properties of the data and the frequency threshold. The useful range of thresholds varies enormously among different datasets. In many applications – such as ▶ basket analysis – the number $|\mathcal{I}|$ of different items can be in thousands, even millions, while the typical sizes of examples are at most in dozens. In such sparse datasets a relatively small number of frequent itemsets can reveal the most outstanding co-occurrences; e.g., there are not likely to be very large sets of books typically bought by the same customers. In dense datasets, in turn, the number of frequent patterns can be overwhelming and also relatively uninformative. E.g., consider the dense dataset of books that have *not* been purchased by a customer: there are a huge number of sets of books that have not been bought by the same customers.

Theory/Solutions

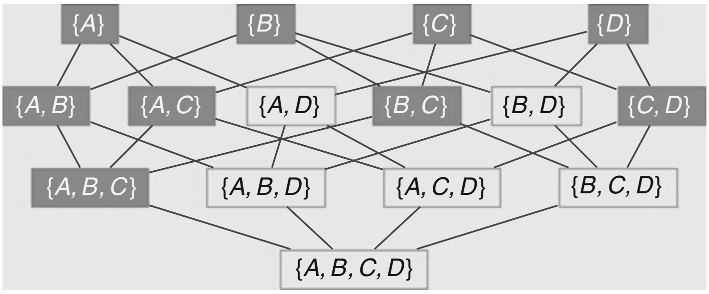
The most widely known solution for finding all frequent itemsets is the ▶ Apriori algorithm (Agrawal et al. 1996). It is based on the monotonicity of itemset frequencies (a ▶ generalization relation): the frequency of a set is at most as high as the frequency of any of its subsets. Conversely, if a set is known to be infrequent, then none of its supersets can be frequent.

Apriori views the ▶ hypothesis space of itemsets as a (refinement) lattice defined by set containment, and performs a ▶ general-to-specific search using breadth-first search. In other words, it starts with singleton itemsets, the most general and frequent sets, and proceeds to larger and less frequent sets. The search is pruned whenever a set does not reach the frequency threshold: all supersets of such sets are excluded from further search. Apriori deviates from standard breadth-first search by evaluating all sets of equal size in a single batch, i.e., it proceeds in a levelwise manner. This has no effect on the search structure or results, but can reduce disk access considerably for large databases. See the entry ▶ Apriori Algorithm for an outline of the method.

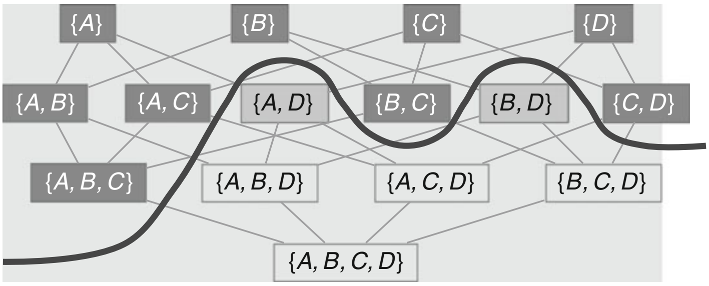
Example 2 Figure 1 illustrates the search space for the data $\mathcal{D}$ of Example 1. Dark nodes represent frequent itemsets, i.e., the answer to the frequent itemset mining problem. Apriori traverses the space a level at a time. For instance, on the second level, it finds out that $\{A, D\}$ and $\{B, D\}$ are not frequent. It therefore prunes all their supersets, i.e., does not evaluate sets $\{A, B, D\}$, $\{A, C, D\}$, and $\{B, C, D\}$ on the third level.

Other search strategies have also been applied. A depth-first search without the subset check allows faster identification of candidates, at the expense of having more candidates to evaluate and doing that without natural batches (e.g. Zaki 2000). FP-growth (Han et al. 2004) uses a tree structure to store the information in the dataset, and uses it to recursively search for frequent itemsets.

The search strategy of Apriori is optimal in a certain sense. Consider the number of sets evaluated, and assume that for any already evaluated set we know whether it was frequent or not but do not consider its frequency. Apriori evaluates the frequencies of all frequent itemsets plus a number of candidates that turn out to be infrequent. It



Frequent Pattern, Fig. 1 The search space of frequent itemsets for data D of the running example. *Dark nodes*: frequent itemsets; *white nodes*: infrequent itemsets



Frequent Pattern, Fig. 2 The positive border ($\{A, B, C\}$, $\{C, D\}$) and negative border ($\{A, D\}$, $\{B, D\}$) of frequent itemsets

turns out that every infrequent candidate must actually be evaluated under the given assumptions: knowing which other sets are frequent and which are not does not help, regardless of the search order. This observation leads to the concept of *border*: the border consists of all those itemsets whose all proper subsets are frequent and whose all proper supersets are infrequent (Gunopulos et al. 2003; Mannila and Toivonen 1997). The border can further be divided into two: the positive border contains those itemsets in the border that are frequent, the negative border contains those that are not. The positive border thus consists of the most specific patterns that are frequent, and corresponds to the "S" set of ► version spaces.

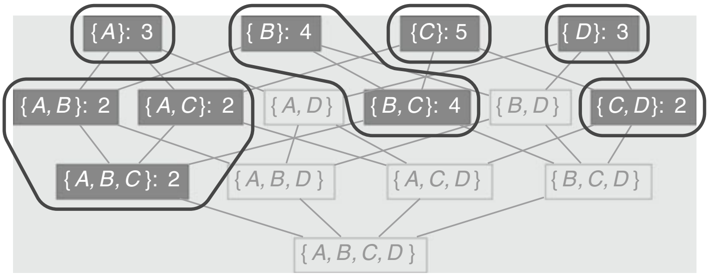
Example 3 Continuing our running example, Fig. 2 illustrates the border between the frequent and infrequent sets. Either the positive or the negative border can alone be used to specify the collection of frequent itemsets: every frequent itemset is a subset of a set in the positive border ($\{A, B, C\}$, $\{C, D\}$), while every infrequent itemset is a superset of a set in the negative border ($\{A, D\}$, $\{B, D\}$).

One variant of frequent itemset mining is to output the positive border only, i.e., to find the *maximal frequent itemsets* (Bayardo 1998). This can be implemented with search strategies that do not need to evaluate the whole space of frequent patterns. This can be useful especially if the number of frequent itemsets is very large, or if the maximal frequent itemsets are large (in which case the number of frequent itemsets is large, too, since the number of subsets is exponential in the length of the maximal set). As a trade-off, the result does not directly indicate frequencies of itemsets.

Condensed Representations: Closed Sets and Nonderivable Sets

Closed sets and nonderivable sets are a powerful concept for working with frequent itemsets, especially if the data is relatively dense or there are strong dependencies. Unlike the aforementioned simple model for borders, here also the known frequencies of sets are used to make inferences about frequencies of other sets.

As a motivation for closed sets (Pasquier et al. 1999), consider a situation where the frequency of itemset $\{i, j\}$ equals the frequency of item j. This implies that whenever j occurs, so does i. Thus, any set $A \cup \{j\}$ that contains item j also contains item i, and the frequencies of sets $A \cup \{j\}$ and $A \cup \{i, j\}$ must be equal. As a result,



Frequent Pattern, Fig. 3 Frequencies and equivalence classes of frequent itemsets in data $\mathcal{D}$ of the running example, and the corresponding closed sets and generators

it sufficies to evaluate sets $A\cup\{j\}$ to obtain the frequencies of sets $A\cup\{i, j\}$, too.

More formally, the *closure* of set A is its largest superset with identical frequency. A is *closed* iff it is its own closure, i.e., if every proper superset of A has a smaller frequency than A. The utility of closed sets comes from the fact that frequent closed sets and their frequencies are a sufficient representation of all frequent sets. Namely, if B is a frequent set then its closure is a frequent closed set in $\mathcal{C}l$, where $\mathcal{C}l$ denotes the collection of all frequent closed itemsets. B's frequency is obtained as $\mathrm{fr}(B) = \max\{\mathrm{fr}(A) | A \epsilon \mathcal{C}l \text{ and } B \subseteq A\}$. If B is not a frequent set, then it has no superset in $\mathcal{C}l$. ▶ Formal concept analysis studies and uses closed sets and other related concepts.

Generators are a complementary concept, and also constitute a sufficient representation of frequent itemsets. (To be more exact, in addition to frequent generators, generators in the border are also needed). Set A is a *generator* (also known as a key pattern or a free set) if all its proper subsets have a larger frequency than A has. Thus, in an equivalence class of itemsets, defined by the set of examples in which they occur, the maximal element is unique and is the closed set, and the minimal elements are generators. The property of being a generator is monotone in the same way that being frequent is, and generators can be found with simple modifications to the Apriori algorithm.

Example 4 Figure 3 illustrates the equivalence classes of itemsets by circles. For instance, the closure of itemset $\{A, B\}$ is $\{A, B, C\}$, i.e., whenever $\{A, B\}$ occurs in the data, C also occurs, but no other items. Given just the frequent closed sets and their frequencies, the frequency of, say, $\{B\}$ is obtained by finding its smallest frequent closed superset. It is $\{B, C\}$, with frequency 4, which is also B's frequency. Alternatively, using generators as the condensed representation, the frequency of itemset $\{B, C\}$ can be obtained by finding its maximal generator subset, i.e., $\{B\}$, with which it shares the same frequency.

Nonderivability of an itemset (Calders and Goethals 2002) is a more complex but often also a more powerful concept than closed sets. Given the frequencies of (some) subsets of itemset A, the frequency of A may actually be uniquely determined, i.e., there is only one possible consistent value. A practical method of trying to determine the frequency is based on deriving upper and lower bounds with inclusion–exclusion formula from the known frequencies of some subsets, and checking if these coincide. An itemset is derivable if this is indeed the case, otherwise it is *nonderivable*. Obviously, the collection of nonderivable frequent sets is a sufficient representation for all frequent sets.

Bounds for the absolute frequency of set I are obtained from its subsets as follows, for any $X \subseteq I$:

$$\mathrm{fr}(I) \le \sum_{J:X\subseteq J\subset I} (-1)^{|I\setminus J|+1}\mathrm{fr}(J) \qquad \text{if}|I\setminus X|\text{is odd}, \tag{1}$$

$$\mathrm{fr}(I) \ge, \sum_{J:X\subseteq J\subset I} (-1)^{|I\setminus J|+1}\,\mathrm{fr}(J) \qquad \text{if } |I\setminus X| \text{ is even}. \tag{2}$$

Using all subsets X of I, one can obtain a number of upper and lower bounds. If the least upper bound equals the greatest lower bound, then set I is derivable. The conceptual elegance of this solution lies in the fact that derivable sets follow logically from the nonderivable ones – the aforementioned formula is one way of finding (some) such situations – whereas with closed sets the user must know the closure properties.

Generalizations of Frequent Patterns

The concept of frequent patterns has been extended in two largely orthogonal directions. One is to more complex patterns and data, such as frequent sequences, trees (see ▶ tree mining), graphs (see ▶ graph mining), and first-order logic (Dehaspe and Toivonen 1999). The other direction to generalize the concept is to ▶ constraint-based mining, where other and more complex conditions are considered beyond frequency. We encourage the interested reader to continue at the entry for ▶ constraint-based mining, which also gives further insight into many of the more theoretical aspects of frequent pattern mining.

Programs and Data

Frequent itemset mining implementations repository: http://fimi.cs.helsinki.fi/
Weka: http://www.cs.waikato.ac.nz/ml/weka/
Christian Borgelt's implementations: http://www.borgelt.net/software.html
Data mining template library: http://dmtl.sourceforge.net/

Applications

Frequent patterns are a general purpose tool for data exploration, with applications virtually everywhere. Market ▶ basket analysis was the first application, telecom alarm correlation and gene mapping are examples of quite different application fields.

Future Directions

Work on frequent pattern mining is being expanded in several directions. New types of pattern languages are being developed, either to meet some specific needs or to increase the expressive power. Many of these developments are motivated by different types of data and applications. Within machine learning, frequent patterns are increasingly being used as a tool for feature construction in complex domains. For an end-user application, methods for choosing and ranking the most interesting patterns among thousands or millions of them is a crucial problem, for which there are no perfect solutions (cf. Geng and Hamilton 2006). At the same time, theoretical understanding of the problem and solutions of frequent pattern discovery still has room for improvement.

Cross-References

▶ Apriori Algorithm
▶ Association Rule
▶ Basket Analysis
▶ Constraint-Based Mining
▶ Frequent Itemset
▶ Graph Mining
▶ Tree Mining

Recommended Reading

Agrawal R, Imielinski T, Swami A (1993) Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD international conference on management of data, Washington, DC. ACM, New York, pp 207–216

Agrawal R, Mannila H, Srikant R, Toivonen H, Verkamo AI (1996) Fast discovery of association rules. In: Fayyad UM, Piatetsky-Shapiro G, Smyth P, Uthurusamy R (eds) Advances in knowledge discovery and data mining. AVAAI Press, Menlo Park, pp 307–328

Bayardo RJ Jr (1998) Efficiently mining long patterns from databases. In: Proceedings of the 1998 ACM SIGMOD international conference on management of data, Seatle, Washington, DC. ACM, New York, pp 85–93

Calders T, Goethals B (2002) Mining all non-derivable frequent itemsets. In: Proceedings of the 6th European conference on principles of data mining and knowledge discovery, Helsinki. Lecture Notes in Computer Science, vol 2431. Springer, London, pp 74–85

Ceglar A, Roddick JF (2006) Association mining. ACM Comput Surv 38(2):5

Dehaspe L, Toivonen H (1999) Discovery of frequent datalog patterns. Data Min Knowl Discov 3(1):7–36

Geng L, Hamilton HJ (2006) Interestingness measures for data mining: a survey. ACM Comput Surv 38(3):9

Gunopulos D, Khardon R, Mannila H, Saluja S, Toivonen H, Sharma RS (2003) Discovering all most specific sentences. ACM Trans Database Syst 28(2):140–174

Han J, Pei J, Yin Y, Mao R (2004) Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min Knowl Discov 8(1):53–87

Mannila H, Toivonen H (1997) Levelwise search and borders of theories in knowledge discovery. Data Min Knowl Discov 1(3):241–258

Pasquier N, Bastide Y, Taouil R, Lakhal L (1999) Discovering frequent closed itemsets for association rules. In: Proceedings of 7th international conference on database theory, Jerusalem. Lecture notes in computer science, vol 1540, pp 398–416. Springer, London

Zaki MJ (2000) Scalable algorithms for association mining. IEEE Trans Knowl Data Eng 12(3):372–390

Frequent Set

▸ Frequent Itemset

Functional Trees

▸ Model Trees

Fuzzy Sets

Fuzzy sets were introduced by Lofti Zadeh as a generalization of the concept of a regular set. A fuzzy set is characterized by a membership function that assigns a degree (or grade) of membership to all the elements in the universe of discourse. The membership value is a real number in the range [0, 1], where 0 denotes no definite membership, 1 denotes definite membership, and intermediate values denote partial membership to the set. In this way, the transition from nonmembership to membership in a fuzzy set is gradual and not abrupt like in a regular set, allowing the representation of imprecise concepts like "small," "cold," "large," or "very" for example.

A variable with its values defined by fuzzy sets is called a linguistic variable. For example, a linguistic variable used to represent a temperature can be defined as taking the values "cold," "comfortable," and "warm," each one of them defined as a fuzzy set. These linguistic labels, which are imprecise by their own nature, are, however, defined very precisely by using fuzzy set concepts.

Based on the concepts of fuzzy sets and linguistic variables, it is possible to define a complete fuzzy logic, which is an extension of the classical logic but appropriate to deal with approximate knowledge, uncertainty, and imprecision.

Recommended Reading

Zadeh LA (1965) Fuzzy sets. Inf Control 8(3):338–353

Fuzzy Systems

A fuzzy system is a computing framework based on the concepts of the theory of ▸ fuzzy sets, fuzzy rules, and fuzzy inference. It is structured in four main components: a knowledge base, a fuzzification interface, an inference engine, and a defuzzification interface. The knowledge base consists of a rule base defined in terms of fuzzy rules, and a database that contains the definitions of the linguistic terms for each input and output linguistic variable. The fuzzification interface transforms the (crisp) input values into fuzzy values, by computing their membership to all linguistic terms defined in the corresponding input domain. The inference engine performs the fuzzy inference process, by computing the activation degree and the output of each rule. The defuzzification interface computes the (crisp) output values by combining the output of the rules and performing a specific transformation.

Fuzzy systems can be classified in different categories. The most widely used are the Mamdani and the Takagi-Sugeno models. In a Mamdani fuzzy system the output variables are defined as linguistic variables while in a Takagi-Sugeno fuzzy system they are defined as a linear combination of the input variables.

Fuzzy systems can model nonlinear functions of arbitrary complexity, however, their main strength comes from their ability to represent imprecise concepts and to establish relations between them.

Recommended Reading

Mamdani EH, Assilian S (1975) An experiment in linguistic synthesis with a fuzzy logic controller. Int J Man-Mach Stud 7(1):1–13

Sugeno M (1985) Industrial applications of fuzzy control. Elsevier Science Publishers, New York

G

Gaussian Distribution

Xinhua Zhang
NICTA, Australian National University,
Canberra, ACT, Australia
School of Computer Science, Australian
National University, Canberra, ACT, Australia
NICTA London Circuit, Canberra, ACT,
Australia

Abstract
Gaussian distributions are one of the most important distributions in statistics. It is a continuous probability distribution that approximately describes some mass of objects that concentrate about their mean. The probability density function is bell shaped, peaking at the mean. Its popularity also arises partly from the central limit theorem, which says the average of a large number of independent and identically distributed random variables is approximately Gaussian distributed. Moreover, under some reasonable conditions, posterior distributions become approximately Gaussian in the large data limit. Therefore, the Gaussian distribution has been used as a simple model for many theoretical and practical problems in statistics, natural science, and social science.

Synonyms

Normal distribution

Definition

The simplest form of Gaussian distribution is the one-dimensional standard Gaussian distribution, which can be described by the probability density function (*pdf*):

$$p(x) = \phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2},$$

where $\frac{1}{\sqrt{2\pi}}$ ensures the normalization, i.e., $\int_{\mathbb{R}} p(x)\mathrm{d}x = 1$. This distribution centers around $x = 0$, and the rate of decay or "width" of the curve is 1.

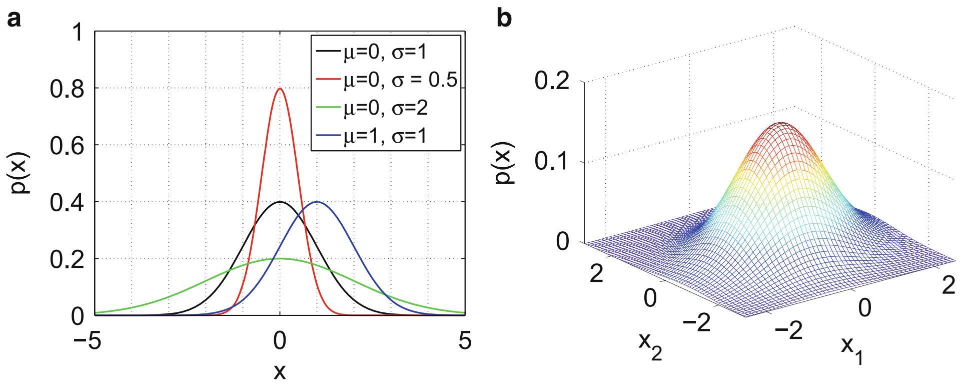
More generally, we can apply translation and scaling to obtain a Gaussian distribution that centers on arbitrary $\mu \in \mathbb{R}$ and with arbitrary width $\sigma > 0$. The *pdf* is

$$\begin{aligned} p(x) &= \frac{1}{\sigma}\phi\left(\frac{x-\mu}{\sigma}\right) \\ &= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \end{aligned}$$

Technically, μ is called the mean and σ^2 is called the variance. Obviously, μ is the peak/mode of the density and is also the mean and median of the distribution due to the symmetry of the density around μ. If a random variable X has this density, then we write

$$X \sim \mathcal{N}(\mu, \sigma^2).$$

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1



Gaussian Distribution, Fig. 1 Gaussian probability density functions. (**a**) One dimension. (**b**) Two dimension

Example density functions are plotted in Fig. 1a.

As an extension to multivariate random variables, the multivariate Gaussian distribution is a distribution on d-dimensional column vector $\mathbf{x}$ with mean column vector $\boldsymbol{\mu}$ and positive definite variance matrix Σ. This gives

$$p(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}\det^{1/2}\Sigma}\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\top}\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right),$$

and is denoted by $X \sim \mathcal{N}(\boldsymbol{\mu},\Sigma)$. An example *pdf* for the two-dimensional case is plotted in Fig. 1b.

Motivation and Background

In history, Abraham de Moivre first introduced this distribution in 1733 under the name "normal distribution" (of course, he did not call it Gaussian distribution since Gauss had not yet been born). Then Laplace used it to analyze experiment errors, based on which Legendre invented the least squares in 1805. Carl Friedrich Gauss rigorously justified it in 1809 and determined the formula of its probability density function. Finally this distribution is named the Gaussian distribution after Gauss. The name "normal distribution" is also widely used, meaning it is a typical, common, or usual distribution. It was coined by Peirce, Galton, and Lexis around 1875 and made popular by Karl Pearson near the inception of the twentieth century.

Theory/Solution

Canonical Form

The standard definition allows one to easily read off the moments from the *pdf*. Another useful parameterization is called canonical parameterization:

$$p(\mathbf{x}|\boldsymbol{\eta},\Lambda) = \exp\left(\boldsymbol{\eta}^{\top}\mathbf{x} - \frac{1}{2}\mathbf{x}^{\top}\Lambda\mathbf{x} - \frac{1}{2}\left(d\log(2\pi) - \log\det\Lambda + \boldsymbol{\eta}^{\top}\Lambda\boldsymbol{\eta}\right)\right),$$

where $\eta = \Sigma^{-1}\boldsymbol{\mu}$ and $\Lambda = \Sigma^{-1}$. Λ is often called precision. This parameterization is useful when posing the distribution as a member of the exponential family.

Cumulative Distribution Function

For one-dimensional Gaussian distribution, the cumulative distribution function (*cdf*) is defined by

$$\Phi(x) = \int_{-\infty}^{x}\phi(t)\mathrm{d}t.$$

Formally, it can be conveniently represented by the error function and its complement:

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \mathrm{d}t,$$

$$\mathrm{erfc}(x) = 1 - \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} \mathrm{d}t.$$

So

$$\Phi(x) = \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right) = \frac{1}{2}\mathrm{erfc}\left(-\frac{x}{\sqrt{2}}\right).$$

The inverse of the *cdf*, called quantile function, can be written as

$$\Phi^{-1}(s) = \sqrt{2}\mathrm{erf}^{-1}(2s-1), \quad \text{for } s \in (0,1).$$

The *cdf* error function erf() and its inverse $\mathrm{erf}^{-1}()$ do not usually have a closed form and can be computed numerically by functions like `ERF` in Fortran and `double erf(double x)` in C/C++. For the multivariate case, the corresponding *cdf* is highly challenging to compute numerically.

Moments

The first order moment is $\mathbb{E}[X] = \boldsymbol{\mu}$, the variance is $\mathrm{Var}[X] = \Sigma$, and all higher order cumulants are 0. Any central moments with odd terms are 0, i.e., $\mathbb{E}[\Pi_{i=1}^d (x_i - \mu_i)^{p_i}] = 0$ when $\sum_i p_i$ is odd.

Entropy and Kullback-Leibler Divergence

The differential entropy of multivariate Gaussian is

$$h(p) = -\int_{\mathbb{R}^d} p(\mathbf{x}) \ln p(\mathbf{x}) \mathrm{d}\mathbf{x}$$
$$= \frac{1}{2} \ln\left((2\pi e)^d \det \Sigma\right).$$

The Kullback-Leibler divergence from $\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1)$ to $\mathcal{N}(\boldsymbol{\mu}_2, \Sigma_2)$ is

$$\mathrm{KL}(\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1) || \mathcal{N}(\boldsymbol{\mu}_2, \Sigma_2))$$
$$= \frac{1}{2}\left(\ln \frac{\det \Sigma_2}{\det \Sigma_1} + \mathrm{tr}\Sigma_2^{-1}\Sigma_1 \right.$$
$$\left. + (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^\top \Sigma_2^{-1} (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) - d \right).$$

Properties Under Affine Transform

Let $X \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$. Suppose A is a linear transform from $\mathbb{R}^d$ to $\mathbb{R}^s$ and $\boldsymbol{c} \in \mathbb{R}^s$, then

$$A\mathbf{x} + \boldsymbol{c} \sim \mathcal{N}(A\boldsymbol{\mu} + \boldsymbol{c}, A\Sigma A^\top)$$
$$\mathbb{E}[(\mathbf{x} - \boldsymbol{\mu})^\top A(\mathbf{x} - \boldsymbol{\mu})] = \mathrm{tr}A\Sigma$$
$$\mathrm{Var}[(\mathbf{x} - \boldsymbol{\mu})^\top A(\mathbf{x} - \boldsymbol{\mu})] = 2\mathrm{tr}A\Sigma A\Sigma$$

where the last two relations require $s = d$.

Conjugate Priors

Conjugate priors where discussed in `<the entry on Prior Probabilities>` (Springer formatters, we want to reference this entry in-line, please format appropriately.). With known variance, the conjugate prior for the mean is again a multivariate Gaussian. With known mean, the conjugate prior for the variance matrix is the Wishart distribution, while the conjugate prior for the precision matrix is the Gamma distribution.

Parameter Estimation

Given n *iid* observations $X_1, \ldots, X_n$, the maximum likelihood estimator of the mean is simply the sample mean

$$\tilde{\boldsymbol{\mu}} = \bar{X} = \frac{1}{n}\sum_{i=1}^n X_i.$$

The maximum likelihood estimator of the covariance matrix is

$$\tilde{\Sigma} = \frac{1}{n}\sum_{i=1}^n (X_i - \bar{X})(X_i - \bar{X})^\top.$$

This estimator is biased, and its expectation is $\mathbb{E}[\tilde{\Sigma}] = \frac{n-1}{n}\Sigma$. An unbiased estimator is

$$\tilde{\Sigma} = S = \frac{1}{n-1}\sum_{i=1}^n (X_i - \bar{X})(X_i - \bar{X})^\top.$$

Distributions Induced by the Gaussian

If $X \sim \mathcal{N}(0, \Sigma)$, then $X^\top \Sigma^{-1} X$ has a Gamma distribution Gamma$(d/2, 2)$.

Let $X_1, X_2 \sim \mathcal{N}(0,1)$ and they are independent. Their ratio is the standard Cauchy distribution, $X_1/X_2 \sim \text{Cauchy}(0,1)$.

Given n independent univariate random variables $X_i \sim \mathcal{N}(0,1)$, the random variable $Z := \sqrt{\sum_i X_i^2}$ has a χ distribution with degree of freedom n. And Z^2 has a χ^2 distribution with degree of freedom n.

Using Basu's theorem or Cochran's theorem, one can show that the sample mean of $X_1, \ldots, X_n$ and the sample standard deviation are independent. Their ratio

$$t := \frac{\bar{X}}{S} = \frac{\frac{1}{n}(X_1 + \ldots + X_n)}{\sqrt{\frac{1}{n-1}\left[(X_1 - \bar{X})^2 + \ldots + (X_n - \bar{X})^2\right]}}$$

has the student's t-distribution with degree of freedom $n-1$.

Applications

This section discusses some applications and properties of the Gaussian.

Central Limit Theorem

Given n independent and identically distributed observations drawn from a distribution whose variance is finite, the average of the observations is asymptotically Gaussian distributed when n tends to infinity. Under certain conditions, the requirement for identical distribution can be relaxed, and asymptotic normality still holds.

Approximate Gaussian Posterior

Consider n independent and identically distributed observations drawn from a distribution $p(X_i|\boldsymbol{\theta})$, so the data set is $\mathbf{X} = (X_1, \ldots, X_n)^\top$. Under certain conditions, saying roughly that the posterior on $\boldsymbol{\theta}$ converges in probability to a single interior point in its domain as $n \to \infty$, the posterior for $\boldsymbol{\theta}$ is approximately Gaussian for large n, $\boldsymbol{\theta}|\mathbf{X} \approx \mathcal{N}\left(\widehat{\boldsymbol{\theta}}, I\left(\widehat{\boldsymbol{\theta}}\right)\right)$, where $\widehat{\boldsymbol{\theta}}$ is the maximum likelihood or aposterior value for $\boldsymbol{\theta}$ and $I(\boldsymbol{\theta})$ is the *observed (Fisher) information*, the negative of the second derivative (Hessian) of the likelihood w.r.t. the parameters $\boldsymbol{\theta}$.

The Gaussian approximation to the posterior, while a poor approximation in many cases, serves as a useful insight into the nature of asymptotic reasoning. It is justified based on the multidimensional Taylor expansion of the log likelihood at the maximum likelihood or a posterior value, together with its asymptotic convergence property.

3-σ Rule

For standard Gaussian distribution, 99.7 % of the probability mass lie within the three standard deviations $[-3\sigma, 3\sigma]$, i.e., $\int_{-3\sigma}^{3\sigma} \phi(x)\mathrm{d}x > 0.997$. About 95 % mass lies within two standard deviations and about 68 % within one standard deviation. This empirical rule is called 3-σ rule and can be easily extended to general one-dimensional Gaussian distributions.

Combination of Random Variables

Let d-dimensional random variables $X_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$. If they are independent, then for any set of linear transforms A_i from $\mathbb{R}^d$ to $\mathbb{R}^s$, we have $\sum_i A_i X_i \sim \mathcal{N}(\sum_i A_i \boldsymbol{\mu}_i, \sum_i A_i \Sigma_i A_i^\top)$. The converse is also true by the Cramer's theorem: if X_i are independent and their sum $\sum_i X_i$ is Gaussian distributed, then all X_i must be Gaussian.

Correlations and Independence

In general, independent random variables must be uncorrelated but not vice versa. However, if a multivariate random variable is jointly Gaussian, then any uncorrelated subset of the random variables *must be* independent. Notice the precondition of joint Gaussian. It is possible for two Gaussian random variables to be uncorrelated but not independent, for the reason that they are not jointly Gaussian. For example, let $X \sim \mathcal{N}(0,1)$ and $Y = -X$ if $|X| < c$, and $Y = X$ if $|X| > c$. By properly setting c, Y and X can be made uncorrelated but obviously not independent.

Marginalization, Conditioning, and Agglomeration

Suppose the vector $\mathbf{x}$ can be written as $(\mathbf{x}_1^\top, \mathbf{x}_2^\top)^\top$ and correspondingly the mean and covariance matrix can be written as

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}$$

Then the marginal distribution of $\mathbf{x}_1$ is Gaussian $\mathcal{N}(\mu_1, \Sigma_{11})$, and the conditional distribution of $\mathbf{x}_1$ conditioned on $\mathbf{x}_2$ is $\mathcal{N}(\boldsymbol{\mu}_{1|2}, \Sigma_{1|2})$, where

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 + \Sigma_{12}\Sigma_{22}^{-1}(\mathbf{x}_2 - \boldsymbol{\mu}_2),$$
$$\Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}.$$

Suppose the multivariate Gaussian vector $\mathbf{x}_1 \sim \mathcal{N}(\mu_1, \Sigma_{11})$ and a vector $\mathbf{x}_2$ is a linear function of $\mathbf{x}_1$ with Gaussian noise, i.e., $\mathbf{x}_2|\mathbf{x}_1 \sim \mathcal{N}(A\mathbf{x}_1 + \boldsymbol{\mu}_{12}, \Sigma_{12})$. Then the joint distribution of $(\mathbf{x}_1^\top, \mathbf{x}_2^\top)^\top$ is also Gaussian:

$$\begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu}_1 \\ A\boldsymbol{\mu}_1 + \boldsymbol{\mu}_{12} \end{pmatrix}, \begin{pmatrix} \Sigma_{11} + A^\top\Sigma_{12}A & -A^\top\Sigma_{12} \\ -\Sigma_{12}A & \Sigma_{12} \end{pmatrix}\right).$$

Cross-References

▶ Gaussian Processes

Recommended Reading

For a complete treatment of Gaussian distributions from a statistical perspective, see Casella and Berger (2002), and Mardia et al. (1979) provides details for the multivariate case. Bernardo and Smith (2000) shows how Gaussian distributions can be used in the Bayesian theory. Bishop (2006) introduces Gaussian distributions in Chap. 2 and shows how it is extensively used in machine learning. Finally, some historical notes on Gaussian distributions can be found at Miller et al., especially under the entries "NORMAL" and "GAUSS."

Bernardo JM, Smith AFM (2000) Bayesian theory. Wiley, Chichester/New York

Bishop C (2006) Pattern recognition and machine learning. Springer, New York

Casella G, Berger R (2002) Statistical inference, 2nd edn. Duxbury, Pacific Grove

Mardia KV, Kent JT, Bibby JM (1979) Multivariate analysis. Academic Press, London/New York

Miller J, Aldrich J et al. Earliest known uses of some of the words of mathematics. http://jeff560.tripod.com/mathword.html

Gaussian Process

Novi Quadrianto[1], Kristian Kersting[2,3], and Zhao Xu[4]
[1]Department of Informatics, SMiLe CLiNiC, University of Sussex, Brighton, UK
[2]Technische Universität Dortmund, Dortmund, Germany
[3]Knowledge Discovery, Fraunhofer IAIS, Sankt Augustin, Germany
[4]Fraunhofer IAIS, Sankt Augustin, Germany

Synonyms

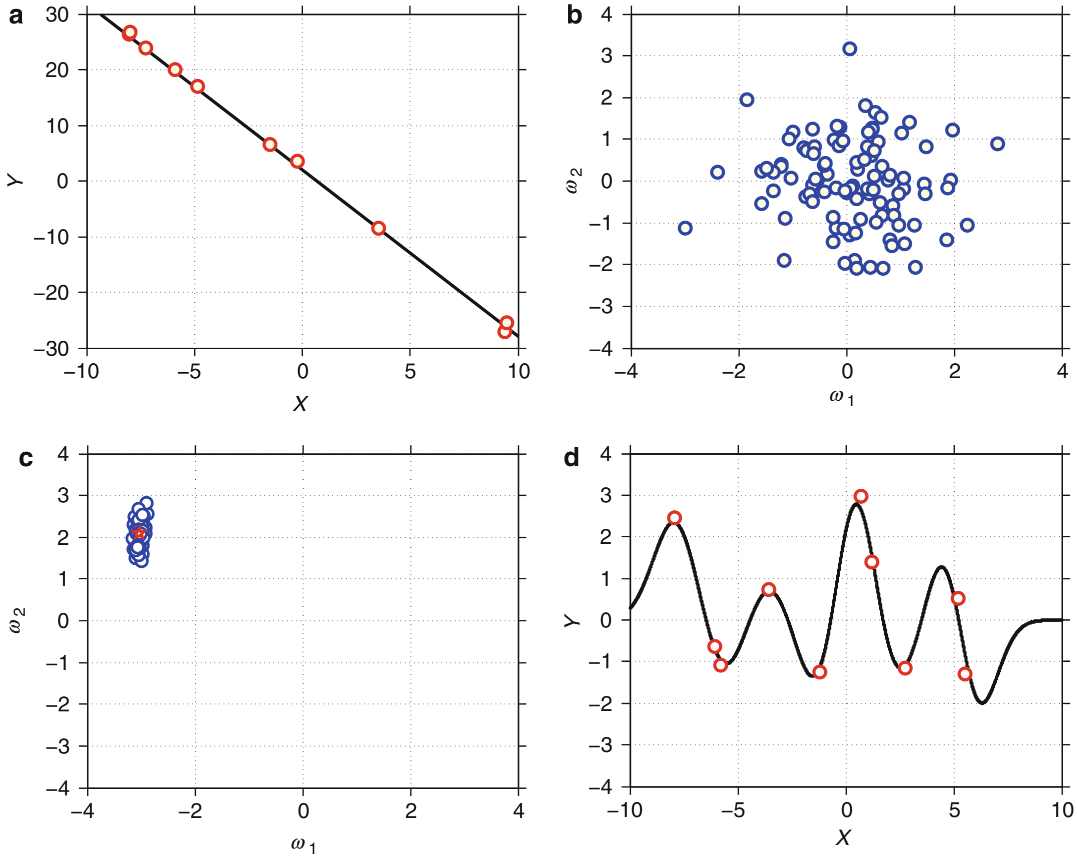
Expectation propagation; Kernels; Laplace estimate; Nonparametric Bayesian

Definition

Gaussian processes generalize multivariate Gaussian distributions over finite-dimensional vectors to infinite dimensionality. Specifically, a Gaussian process is a stochastic process that has Gaussian-distributed finite-dimensional marginal distributions, hence the name. In doing so, it defines a distribution over functions, i.e., each draw from a Gaussian process is a function. Gaussian processes provide a principled, practical, and probabilistic approach to inference and learning in kernel machines.

Motivation and Background

Bayesian probabilistic approaches have many virtues, including their ability to incorporate prior knowledge and their ability to link related



Gaussian Process, Fig. 1 (**a**) Ten observations (one-dimensional input x and output y variables) generated from a ▶ Linear Regression model $y = -3x + 2 + \epsilon$ with Gaussian noise ϵ. The task is to learn the functional relationship between x and y. Assuming the parametric model $y = \omega_1 x + \omega_2 + \epsilon$, i.e., $\omega = (\omega_1, \omega_2)$, is the vector of parameters, and the prior distribution over ω is a two-dimensional Gaussian as shown in (**b**), the posterior distribution over ω can be estimated as shown in (**c**). Its mean $(-2.9716, 1.9981)$ is close to the true parameters $(-3, 2)$. The inference, however, was performed in an ideal situation where in the relationship between x and y was indeed linear. If the true relationship is not known in advances and/or cannot easily be described using a finite set of parameters, this approach may fail. For example, in (**d**), infinite number of parameters might be required to recover the functional relationship

sources of information. Typically, we are given a set of data points sampled from an underlying but unknown distribution, each of which includes input x and output y, such as the ones shown in Fig. 1a. The task is to learn a functional relationship between x and y. Traditionally, in a parametric approach, an assumption on the mathematical form of the relationship such as linear, polynomial, exponential, or a combination of them needs to be chosen a priori. Subsequently, weights (or parameters) are placed on each of the chosen forms, and a prior distribution is then defined over parameters. Thus, the learning task is now reduced to the Bayesian estimation over the parameters, cf. Fig. 1a–c. This approach, however, may not always be practical, as illustrated in Fig. 1d. To discover the latent input–output relationship in Fig. 1d, we might need infinitely many functional forms, and this translates to infinite number of parameters. Instead of working over a parameter space, Gaussian processes place a prior directly on the space of functions without parameterizing the function, hence nonparametric. As will be shown, the computational complexity of inference now scales as the number of data points instead of the number of parameters.

Several nonparametric Bayesian models have been developed for different tasks such as density estimation, regression, classification, survival time analysis, topic modeling, etc. Among the most popular ones are ▶ Dirichlet Processes and *Gaussian processes*. Just as the Gaussian process, a Dirichlet process has Dirichlet-distributed finite-dimensional marginal distributions, hence the name.

Gaussian processes were first formalized for machine-learning tasks by Williams and Rasmussen (1996) and Neal (1996).

Theory

Formally, a Gaussian process is a stochastic process (i.e., a collection of random variables) in which all the finite-dimensional distributions are multivariate Gaussian distributions for any finite choice of variables. In general, Gaussian processes are used to define a probability distribution over functions $f : \mathcal{X} \rightarrow \mathbb{R}$ such that the set of values of f evaluated at an arbitrary set of points $\{x_i\}_{i=1}^{N} \in \mathcal{X}$ will have an N-variate Gaussian distribution. Note that, for $x_i \in \mathbb{R}^2$, this may also be known as a Gaussian random field.

Gaussian Process

A Gaussian distribution is completely specified by its mean and covariance matrix. Similarly, a Gaussian process is characterized by its mean function $m(x) := \mathbf{E}[f(x)]$ and covariance function

$$C(x, x') := \mathbf{E}[(f(x) - m(x))(f(x') - m(x'))] \, .$$

We say a real process $f(x)$ is a Gaussian process distributed with a mean function $m(x)$ and a covariance function $C(x, x')$, written as $f \sim \mathcal{GP}(m(x), C(x, x'))$.

The mean function can be arbitrarily chosen (for convenience, it is often taken to be a zero function since we can always center our observed outputs to have a zero mean), but the covariance function must be a positive-definite function to ensure the existence of all finite-dimensional distributions. That is, the positive definiteness of $C(.,.)$ ensures the positive (semi-)definiteness of all covariance matrices, Σ, appearing in the exponent of the finite-dimensional multivariate Gaussian distribution.

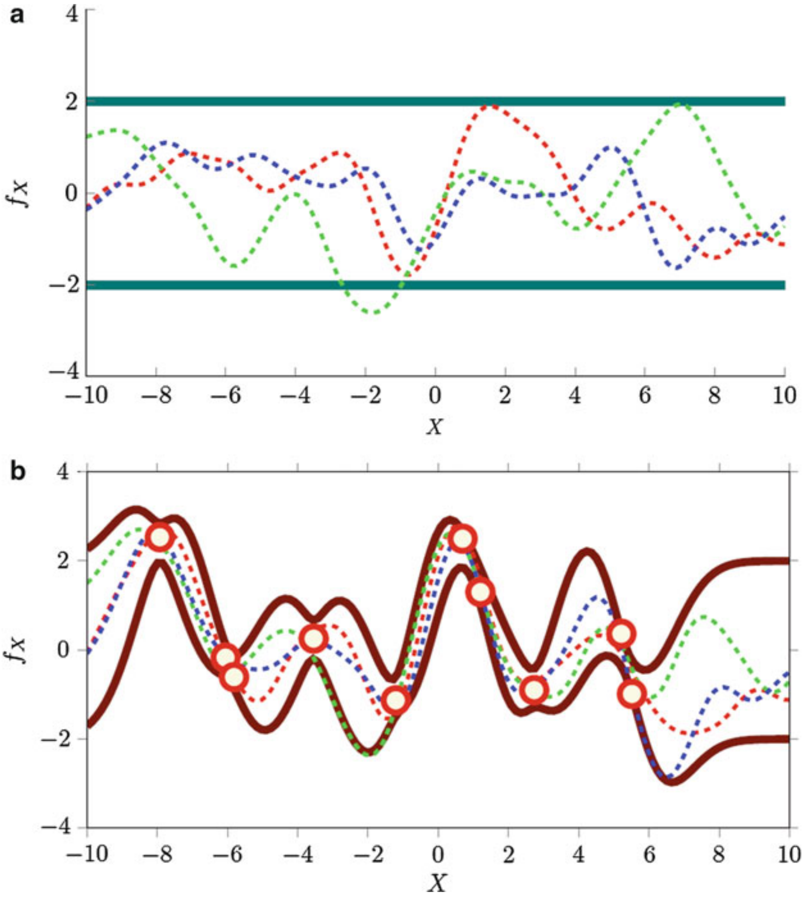
The attractiveness of Gaussian processes is that they admit the marginalization property (▶ Gaussian Distribution), i.e., if the Gaussian process specifies $(f(x_1), \; f(x_2)) \sim \mathcal{N}(\mu, \Sigma)$, then it must also specify $f(x_1) \sim \mathcal{N}(\mu_1, \Sigma_{11})$, where Σ_{11} is the relevant submatrix of Σ. This means addition of novel points will not influence the distribution of existing points. The marginalization property allows us to concentrate on distribution of only observed data points with the rest of unobserved points considered to be marginalized out; thus, a finite amount of computation for inference can be achieved.

Covariance Functions

A covariance function bears an essential role in a Gaussian process model as its continuity properties determine the continuity properties of samples (functions) from the Gaussian process. In the literature, covariance functions are also known as positive (semi-)definite kernels or Mercer's kernels.

There are generally two types of covariance functions: *stationary* and *nonstationary*. A stationary covariance function is a function that is translation invariant, i.e., $C(x, x') = D(x - x')$ for some function D. The typical examples include squared exponential, Matérn class, γ-exponential, exponential, and rational quadratic, while examples of nonstationary covariance functions are dot product and polynomial.

Squared exponential (SE) is a popular form of stationary covariance function, and it corresponds to the class of sums of infinitely many Gaussian-shaped basis functions placed everywhere, $f(x) := \lim_{n\rightarrow\infty} \frac{s}{n} \sum_i^n \gamma_i \exp\left(-((x - x_i)/2\ell)^2\right)$ with $\gamma_i \sim \mathcal{N}(0, 1) \; \forall i$. This covariance function is in the form of



Gaussian Process, Fig. 2 (**a**) Three functions drawn at random from a Gaussian process prior. (**b**) Three random functions drawn from the posterior, i.e., the distribution learned with the prior from Fig. 2a and the ten observations from Fig. 1d. In both plots, the *shaded area* within two *solid lines* shows the pointwise mean plus and minus two times the standard deviation for each input value, i.e., the 95 % confidence region. Animations, if they are visible, are generated using the method described in Hennig (2013)

$$C(x, x') = \mathbf{E}[f(x)f(x')]$$
$$= s^2 \exp\left(-\frac{1}{2\ell^2}\|x - x'\|_2^2\right).$$

Typical functions sampled from this covariance function can be seen in Fig. 2a. This covariance function has the characteristic length scale ℓ and the signal variance s^2 as free parameters (hyperparameters). The longer the characteristic length scale, the more slowly varying the typical sample function is. The signal variance defines the vertical scale of variations of a sample function. Figure 3 illustrates prediction with SE covariance function with varying characteristic length scale. Several other covariance functions are listed in Table 1.

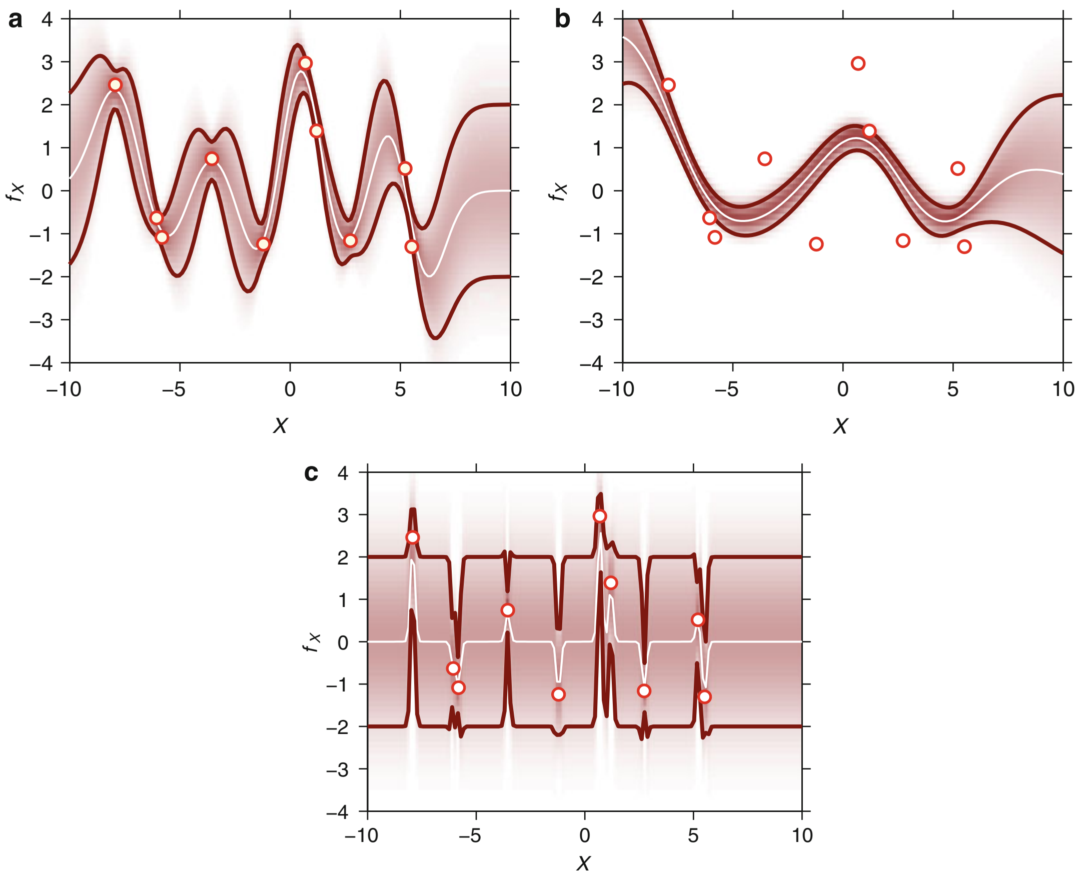
For a comprehensive review on the field of covariance functions, we refer interested readers to Abrahamsen (1992).

Applications

For Gaussian processes, there are two main classes of applications: regression and classification. We will discuss each of them in turn.

Regression

In a ▶ Regression problem, we are interested to recover a functional dependency $y_i = f(x_i) + \epsilon_i$ from N observed training data points $\{(x_i, y_i)\}_{i=1}^N$, where $y_i \in \mathbb{R}$ is the noisy observed output at input location $x_i \in \mathbb{R}^d$. Traditionally, in the Bayesian ▶ Linear Regression model, this regression problem is tackled by requiring us to parameterize the latent function f by a parameter $w \in \mathbb{R}^H$, $f(x) := \langle \phi(x), w \rangle$ for H fixed basis functions $\{\phi_h(x)\}_{h=1}^H$. A prior distribution is then defined over parameter w. The idea of the Gaussian process regression (in the



Gaussian Process, Fig. 3 The Gaussian process prediction with the SE kernel. (**a**) Mean of the prediction distribution with length scale 1.0 and signal variance 1.0 (the hyperparameters of the original process used to generate the data in Fig. 1). The other two plots show the prediction setting of the length scale: **(b)** longer (3.0) and **(c)** shorter (0.1). In all plots, the 95 % confidence region is shown

Gaussian Process, Table 1 Examples of covariance functions. θ_{cov} denotes the set of hyperparameters

Name	$C(x,x')$	θ_{cov}	Remark
Squared exp. (SE)	$s^2 \exp\left(-\frac{1}{2\ell^2}\,\\|x-x'\\|_2^2\right)$	$\{s,\ell\}$	Strong smoothness assumption
Matérn class	$\frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}\lvert x-x'\rvert}{\ell}\right)^{\nu} K_\nu(\frac{\sqrt{2\nu r}}{\ell})$	$\{\nu,\ell\}$	Less smooth than SE
γ-exponential	$\exp(-(\lvert x-x'\rvert/\ell)^{\gamma}),\ \text{with } 0<\gamma<=2$	$\{\ell\}$	Includes both Exp. and SE
Exponential	$\exp\left(\frac{-\lvert x-x'\rvert}{\ell}\right)$	$\{\ell\}$	$\nu = 1/2$ in the Matérn class
Rational quadratic	$\left(1+\frac{\\|x-x'\\|_2^2}{2\alpha\ell^2}\right)^{-\alpha}$	$\{\alpha,\ell\}$	An infinite sum of SE
Dot product	$\sigma_w^2\,\langle x,x'\rangle+\sigma_c^2$	$\{\sigma_w,\sigma_c\}$	
Polynomial	$(\langle x,x'\rangle+\sigma_c^2)^p$	$\{\sigma_c\}$	Effective for high-dimensional classification with binary or grayscale input

geostatistical literature, this is also called *kriging*; see, e.g., Krige 1951; Matheron 1963) is to place a prior directly on the space of functions without parameterizing the function (vide Motivation and Background).

Likelihood Function and Posterior Distribution

Assuming independent and normally distributed noise terms, $\epsilon_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2_{\text{noise}})$, the likelihood model on an output vector $Y \in \mathbb{R}^N$ and an input matrix $X \in \mathbb{R}^{N \times d}$ will be

$$\begin{aligned} p(Y \mid f_X) &= \prod_{i=1}^{N} p(y_i | x_i, f) \\ &= \mathcal{N}(Y \mid f_X, \sigma^2_{\text{noise}} I), \end{aligned}$$

with $f_X = (f(x_1), \ldots, f(x_N))^\top$ be an N-dimensional vector of function values at N input locations x_i. That is, the data likelihood is distributed according to a Gaussian distribution with the function values evaluated at training input locations as its mean and the variance of the noise terms as its variance.

Placing a (zero mean) Gaussian process prior over functions

$$f \sim \mathcal{GP}(m(x) \equiv 0, k(x, x')), \tag{1}$$

will lead us to a Gaussian process posterior (this form of posterior process is described in the next section):

$$\begin{aligned} &f | X, Y \sim \mathcal{GP}(m_{\text{post}}(x) \\ &= k(x, X)[K + \sigma^2_{\text{noise}} I]^{-1} Y, k_{\text{post}}(x, x') \\ &= k(x, x') - k(x, X)[K + \sigma^2_{\text{noise}} I]^{-1} k(x', X)). \end{aligned} \tag{2}$$

In the above equations, $K \in \mathbb{R}^{N \times N}$ denotes the Gram matrix with elements $K_{ij} = k(x_i, x_j)$, and $k(x, x')$ is the kernel function. The term $k(x, X)$ denotes a kernel function with one of the inputs fixed at training points.

Predictive Distribution

The final goal in regression is to make an output prediction for a novel input x_*, given a set of input–output training points. By the marginalization property, instead of working with a prior over infinite-dimensional function spaces as in (1), we can concentrate on the marginal distribution over the training inputs:

$$f_X \sim \mathcal{N}(0, K). \tag{3}$$

Subsequently, the marginal distribution over training outputs (conditioned on inputs) can be computed via

$$\begin{aligned} p(Y | X) &= \int p(Y \mid f_X) p(f_X) \mathrm{d} f_X \\ &= \mathcal{N}(0, K + \sigma^2_{\text{noise}} I). \end{aligned} \tag{4}$$

The above integration is computed by using the standard result for the convolution of two Gaussian distributions (▶ Gaussian Distribution).

Therefore, given inputs X, the joint distribution over outputs Y and the latent function f_X is given by

$$p(Y, f_X | X) = \mathcal{N}(0, C), \tag{5}$$

where $C \in \mathbb{R}^{(2N) \times (2N)}$ is the joint covariance matrix. We can partition this joint covariance matrix as follows:

$$C = \begin{bmatrix} K + \sigma^2_{\text{noise}} I & K \\ K & K \end{bmatrix},$$

The noise variance appears only at the diagonal elements of the covariance matrix C; this is due to the independence assumption about the noise. Using a standard Gaussian property on computing conditional distribution from a joint Gaussian distribution (▶ Gaussian Distribution), the Gaussian posterior distribution can be seen to admit the following form: $p(f_X | X, Y) = \mathcal{N}(\mu_{f_X}, \Sigma_{f_X})$ with the mean $\mu_{f_X} = K(K + \sigma^2_{\text{noise}} I)^{-1} Y$ and the covariance $\Sigma_{f_X} = K - K(K + \sigma^2_{\text{noise}} I)^{-1} K$.

From the posterior distribution, we can compute the predictive distribution on the new output

y^* at an input location x^*, as follows:

$$p(y_*|x_*,X,Y) = \int p(y_*|f_*,x_*)$$
$$\times \int p(f_*|f_X)p(f_X|X,Y)\mathrm{d}f_X\mathrm{d}f_*. \qquad (6)$$

The $p(f_*|f_X)$ is a conditional multivariate Gaussian with mean $\mu_{f_*|f_X} = k_{X,x_*}^\top K^{-1} f_X$ and variance $\sigma^2_{f_*|f_X} = k(x_*,x_*) - k_{X,x_*}^\top K^{-1} k_{X,x_*}$ due to the GP marginalization property, where the vector $k_{X,x_*} \in \mathbb{R}^N$ has elements $k(x_i,x_*)$ for $i = 1,\ldots,N$ and $\top$ denotes a transpose operation. Equation (6) is a general equation for computing the predictive distribution in a Gaussian process framework and is applicable, among others, for both regression and classification settings. For regression, since all the terms are Gaussians, the inner integration $\int p(f_*|f_X)p(f_X|X,Y)\mathrm{d}f_X$ is a Gaussian with mean $\mu_{f_*} = k_{X,x_*}^\top K^{-1}\mu_{f_X}$ and variance $\sigma^2_{f_*} = k(x_*,x_*) - k_{X,x_*}^\top K^{-1} k_{X,x_*} + k_{X,x_*}^\top K^{-1} \Sigma_{f_X} K^{-1} k_{X,x_*}$. Subsequently, the outer integration $\int \mathcal{N}(f_*,\sigma^2_{\text{noise}})\mathcal{N}(\mu_{f_*},\sigma^2_{f_*})\mathrm{d}f_*$ is also a Gaussian, and therefore the above $p(y_*|x_*,X,Y)$ is a Gaussian distribution, and it is in the form of

$$p(y_*|x_*,X,Y) = \mathcal{N}(\mu_*,\sigma^2_*), \qquad (7)$$

with

$$\mu_* = k_{X,x_*}^\top (K + \sigma^2_{\text{noise}} I)^{-1} Y, \qquad (8)$$
$$\sigma^2_* = k(x_*,x_*)$$
$$- k_{X,x_*}^\top (K + \sigma^2_{\text{noise}} I)^{-1} k_{X,x_*} + \sigma^2_{\text{noise}}. \qquad (9)$$

Note that (8) and (9) are the mean function and the covariance function of the posterior process in (2) for any novel inputs. The only difference is the additional term σ^2_{noise}, since there exists observation noise ϵ_* such that $y_* = f_* + \epsilon_*$.

Point Prediction

The previous section has shown how to compute a predictive distribution for outputs y_* associated with the novel test inputs x_*. To convert this predictive distribution into a point prediction, we need the notion of a loss function, $\mathcal{L}(y_{\text{true}}, y_{\text{prediction}})$. This function specifies the loss incurred for predicting the value $y_{\text{prediction}}$, while the true value is y_{true}. Thus, the optimal point prediction can be computed by minimizing the expected loss as follows:

$$y_{\text{optimal}}|x_* = \underset{y_{\text{prediction}} \in \mathbb{R}}{\operatorname{argmin}} \int \mathcal{L}(y_*, y_{\text{prediction}})$$
$$\times p(y_*|x_*,X,Y)\mathrm{d}y_*. \qquad (10)$$

For a squared loss function (or any other symmetric loss functions) and predictive distribution (7), the solution to the above equation is the mean of the predictive distribution, i.e.,

$$y_{\text{optimal}}|x_* = \mathbf{E}_{y_* \sim p(y_*|x_*,X,Y)}[y_*] = \mu_*.$$

The above Gaussian process regression description is known as a function space view in the literature (Rasmussen and Williams 2006). Equivalently, a Gaussian process regression can also be viewed from the traditional Bayesian linear regression with a possibly infinite number of basis functions $\phi(x)$ and with a zero mean and arbitrary positive-definite covariance matrix Gaussian prior on the parameter w; see, e.g., Rasmussen and Williams (2006).

Classification

Gaussian process models can also be applied to classification problems. In a probabilistic approach to classification, the goal is to model posterior probabilities of an input point x_i belonging to one of Ω classes, $y_i \in \{1,\ldots,\Omega\}$. These probabilities must lie in the interval [0, 1]; however, a Gaussian process model draws functions that lie on $(-\infty,\infty)$. For the binary classification ($\Omega = 2$), we can turn the output of a Gaussian process into a class probability using an appropriate nonlinear activation function. In the following, we will show this for the case of binary classification. For the more general cases, see, e.g., Rasmussen and Williams (2006).

Likelihood Function and Posterior Distribution

In a regression problem, a Gaussian likelihood is chosen and combined with a Gaussian process prior, which leads to a Gaussian posterior process over functions where in all required integrations remain analytically tractable. For classification, however, Gaussian likelihood is not the most suitable owing to the discreteness nature of the class labels. The most commonly used likelihood functions are

$$\begin{aligned} p(y_i \,|\, f, x_i) &= \frac{1}{1+\exp(-y_i f_{x_i})} \quad \text{or} \\ p(y_i \,|\, f, x_i) &= \int_{-\infty}^{y_i f_{x_i}} \mathcal{N}(0,1)\mathrm{d}t \\ &= \Phi_{0,1}(y_i f_{x_i}), \end{aligned} \tag{11}$$

known as logistic and cumulative Gaussian likelihood functions, respectively. Assuming that the class labels (conditioned on f) are generated independent and identically distributed, the joint likelihood over N data points can be expressed as $p(Y \,|\, f_X) = \prod_{i=1}^{N} p(y_i \,|\, f, x_i)$. By Bayes' rule, the posterior distribution over latent functions is given by $p(f_X|X,Y) = \frac{p(Y \,|\, f_X)p(f_X)}{\int p(Y \,|\, f_X)p(f_X)\mathrm{d}f_X}$. This posterior is no longer analytically tractable (due to intractable integration in the denominator), and an approximation is needed.

There are several approximation methods to handle intractability of the inference stage in the Gaussian process classification such as Laplace approximation, expectation propagation, variational bounding, and MCMC, among others (see Nickisch and Rasmussen (2008) for a comprehensive overview of approximate inference in binary Gaussian process classification). Most of the methods (if not all) approximate the non-Gaussian posterior with a tractable Gaussian distribution. We describe in detail the straightforward Laplace approximation method, but note that the more complicated expectation propagation (▶ Expectation Propagation) is almost always the method of choice unless the computational budget is very tight (Nickisch and Rasmussen 2008).

Laplace method approximates the non-Gaussian posterior with a Gaussian one by performing a second-order Taylor expansion of the log posterior, $\log p(f_X|X,Y)$ at the maximum point of the posterior

$$p(f_X|X,Y) \approx \hat{p}(f_X|X,Y) = \mathcal{N}(\hat{f}_X, H^{-1}), \tag{12}$$

where $\hat{f}_X = \operatorname{argmax}_{f_X} \log p(f_X|X,Y)$ and $H := -\nabla\nabla \log p\ (f_X|X,Y)|_{f_X=\hat{f}_X}$ is the Hessian of the negative log posterior at the maxima. Since the denominator of the Bayes' theorem is independent of the latent function, the mode of the posterior can be computed instead from the log un-normalized posterior:

$$\Psi(f_X) := \log p(Y \,|\, f_X) + \log p(f_X), \tag{13}$$

with the expression for $p(f_X)$ given in (3). Computation of the mode requires us to evaluate the gradient of $\Psi(f_X)$ which is given as

$$\nabla\Psi(f_X) = \nabla \log p(Y \,|\, f_X) - K^{-1} f_X. \tag{14}$$

To find the stationary point, however, we cannot simply set this gradient to zero as $\nabla \log p(Y \,|\, f)$ depends nonlinearly on f_X. We need to resort to an iterative scheme based on the Newton–Raphson's method with the updated equation given by

$$f_X^{\text{new}} \leftarrow f_X^{\text{old}} - (\nabla\nabla\Psi(f_X))^{-1}\nabla\Psi(f_X), \tag{15}$$

and the Hessian given by

$$\nabla\nabla\Psi(f_X) = -W - K^{-1}, \tag{16}$$

and $W := -\nabla\nabla \log p(Y \,|\, f_X)$ is a diagonal matrix. It is important to note that if the likelihood function $p(Y|f_X)$ is log-concave, the diagonal elements of W are nonnegative, and the Hessian in (16) is negative definite (since $-K$ and its inverse are negative definite by construction and the sum of two negative-definite matrices is also negative definite). Thus, $\Psi(f_X)$ is concave and has a unique maxima point.

Predictive Distribution

The latent function f_X plays the role of a nuisance function, i.e., we do not observe values of f_X itself, and more importantly, we are not particularly interested in the values of f_X. What we are interested in is a class conditional posterior probability, $p(y_* = +1|x_*, X, Y)$, for a novel input x_*. We note that a class conditional probability of a class label of not 1 is $p(y_* = -1|x_*, X, Y) = 1 - p(y_* = +1|x_*, X, Y)$.

We use Equation (6) to compute the predictive distribution on the new output y^* at an input location x^*, restated here for the sake of readability:

$$p(y_*|x_*, X, Y) = \int p(y_*|f_*, x_*) \times \int p(f_*|f_X) p(f_X|X, Y) \mathrm{d} f_X \mathrm{d} f_*.$$

As in regression, the term $p(f_*|f_X)$ is a conditional multivariate Gaussian with the assumption that the underlying Gaussian process model is a noise-free process. Another approach would be assuming an independent Gaussian noise in combination with a step function likelihood function. However, this is equivalent to the noise-free latent process with a cumulative Gaussian likelihood function (Rasmussen and Williams 2006). With Laplace approximation of posterior distribution $p(f_X|X, Y) \approx \mathcal{N}(\hat{f}_X, (K^{-1} + W)^{-1})$, we can now compute the inner integration of the predictive distribution, $\int \mathcal{N}(\mu_{f_*|f_X}, \sigma^2_{f*|f_X}) \mathcal{N}(\hat{f}_X, (K^{-1} + W)^{-1}) \mathrm{d} f_X$, by using the standard result for the convolution of two Gaussian distributions. It is again a Gaussian with mean $\mu_{f_*} = k^{\top}_{X,x_*} K^{-1} \hat{f}_X$ and variance $\sigma^2_{f_*} = k(x_*, x_*) - k^{\top}_{X,x_*}(K + W^{-1})^{-1} k_{X,x_*}$.

The predictive distribution can now be computed as follows:

$$\pi_* := p(y_* = +1|x_*, X, Y) = \int p(y_* = +1|f_*, x_*) \mathcal{N}(\mu_{f_*}, \sigma^2_{f_*}) \mathrm{d} f_*.$$

The above integral can be solved analytically for a cumulative Gaussian likelihood function,

$$\pi_* = \int_{-\infty}^{\frac{\mu_{f_*}}{(y_*^{-2}+\sigma^2_{f_*})^{1/2}}} \mathcal{N}(t|0, 1) \mathrm{d}t = \Phi_{0,1}\left(\frac{\mu_{f_*}}{(y_*^{-2} + \sigma^2_{f_*})^{1/2}}\right),$$

and can be approximated for a logistic likelihood function (MacKay 1992),

$$\pi_* = \frac{1}{1 + \exp(-\mu_{f_*} \kappa(\sigma^2_{f_*}))},$$

with $\kappa(c) = (1 + c\pi/8)^{-1/2}$.

Point Prediction

Similar to the regression case, we might need to make a point prediction from the predictive distribution described in the section above. For a zero-one loss function, i.e., a loss of one unit is suffered for a wrong classification and 0 for not making a classification mistake, the optimal point prediction (in the sense of expected loss) is

$$y_{\text{optimal}}|x^* = \operatorname*{argmax}_{y_* \in \{1,\ldots,\Omega\}} \; p(y_*|x_*, X, Y). \quad (17)$$

It is worth noting that the probabilistic approach to classification allows the same inference stage to be reused with different loss functions. In some situations, a cost-sensitive loss function, i.e., different classification mistakes incur different losses, is more desirable. The optimal point prediction is now taken by minimizing expected cost-sensitive loss with respect to the same $p(y_*|x_*, X, Y)$.

Extension of binary classification to multiclass Gaussian process classification ($\Omega > 2$) can be achieved via the softmax activation function, i.e., a generalization of logistic activation function (refer to Williams and Barber (1998) for the Laplace approximation of the posterior distribution). Recently, Bratières, Quadrianto, and Ghahramani (to appear) propose a Gaussian process classification approach to structured output problems ($\Omega \gg 2$) using a generalization of softmax function called a structured softmax function. Examples of structured outputs are a tree, a grid, or a sequence, where the output consists of interdependent categorical atoms.

Practical Issues

We have seen how to do regression and classification using Gaussian processes. Like other kernel-based methods such as support vector machines, they are very flexible in that all operations are kernelized, i.e., the operations are performed in the (possibly infinite dimensional) feature space. However, this feature space is only defined implicitly via positive-definite kernels (covariance functions), which only require computation in the (lower dimensional) input space. Compared to other non-Bayesian kernel approaches, Gaussian processes provide an explicit probabilistic formulation of the model. This directly provides us with confidence intervals (for regression) or posterior class probabilities (for classification).

So far, however, we have assumed a covariance function with the known functional form and hyperparameters. In many practical applications, it may not be easy to specify all aspects of the covariance function by hand. Furthermore, inverting the corresponding $N \times N$ Gram matrix is the main computational cost, and it may be prohibitive as it scales as $\mathcal{O}(N^3)$. We will now discuss approaches to overcome both limitations in turn.

Model Selection

In many practical applications, the functional form of the covariance function needs to be chosen, and any values of hyperparameters associated with the chosen covariance function and possible free parameters of the likelihood function need to be optimally determined. This is called model selection.

Ideally, we would like to define a prior distribution over the hyperparameters θ, and predictions are made by integrating over different possible choices of hyperparameters. More formally,

$$\begin{aligned} & p(y_*|x_*, X, Y) \\ & = \int p(y_*|x_*, X, Y, \theta) p(\theta|X, Y) \mathrm{d}\theta. \end{aligned} \quad (18)$$

The evaluation of the above integral, however, may be difficult, and an approximation is needed either by using the most likely value of hyperparameters, $p(y_*|x_*, X, Y) \approx p(y_*|x_*, X, Y, \theta_{\mathrm{ML}})$, or by performing the integration numerically via Monte Carlo methods. We will focus here on the approximation approach and show how to use it for regression and classification problems.

Marginal Likelihood for Regression

The posterior probability of the hyperparameters θ in (18) is

$$p(\theta|X, Y) \propto p(Y|X, \theta) p(\theta), \quad (19)$$

where the first term is known as marginal likelihood or evidence for the hyperparameters and it is in the form of (as in (4) but with an explicit conditioning on θ)

$$\begin{aligned} p(Y|X, \theta) &= \int p(Y|\, f_X, \theta) p(f_X|\theta) \mathrm{d} f_X \\ &= \mathcal{N}(0, K + \sigma^2_{\text{noise}} I), \end{aligned}$$

where the set of free parameters θ includes both parameters of the kernel function and the noise term σ^2_{noise}. We can then set the hyperparameters by maximizing the logarithm of this marginal likelihood, and its partial derivative with respect to hyperparameters is

$$\begin{aligned} & \frac{\partial}{\partial \theta_j} \log p(Y|X, \theta) \\ & = \frac{1}{2} Y^\top \bar{K}^{-1} \frac{\partial \bar{K}}{\partial \theta_j} \bar{K}^{-1} Y - \frac{1}{2} \mathrm{tr} \left(\bar{K}^{-1} \frac{\partial \bar{K}}{\partial \theta_j} \right), \end{aligned}$$

with $\bar{K} := K + \sigma^2_{\text{noise}} I$. This is known as a type II maximum likelihood approximation, ML-II. We can also maximize the un-normalized posterior instead, assuming finding the derivatives of the priors is straightforward.

Marginal Likelihood for Classification

The Laplace approximation of the marginal likelihood, $p(Y|X, \theta) \approx \hat{p}(Y|X, \theta)$

$$= \int \exp(\Psi(f_X))\mathrm{d}f_X$$
$$= \exp(\Psi(\hat{f}_X)) \int \exp(-\frac{1}{2}(f_X - \hat{f}_X)^\top H(f_X - \hat{f}_X))\mathrm{d}f_X,$$

which is achieved via a Taylor expansion of (13) locally around $\hat{f}_X$ to obtain $\Psi(f_X) \approx \Psi(\hat{f}_X) - \frac{1}{2}(f_X - \hat{f}_X)^\top H(f_X - \hat{f}_X)$. Computing the integral analytically gives us the approximate marginal likelihood

$$\log \hat{p}(Y|X,\theta) \propto -\frac{1}{2}\hat{f}_X K^{-1}\hat{f}_X - \frac{1}{2}\log|K| + \log p(Y|\hat{f}_X) - \frac{1}{2}\log|K^{-1}+W|.$$

Subsequently, the partial derivatives with respect to hyperparameters is given by

$$\frac{\partial}{\partial\theta_j}\log \hat{p}(Y|X,\theta) = \frac{1}{2}\hat{f}_X^\top K^{-1}\frac{\partial K}{\partial\theta_j}K^{-1}\hat{f}_X - \frac{1}{2}\mathrm{tr}\left((K+W^{-1})^{-1}\frac{\partial K}{\partial\theta_j}\right) + \sum_{i=1}^{N}\frac{\partial\log\hat{p}(Y|X,\theta)}{\partial\hat{f}_{x_i}}\frac{\partial\hat{f}_{x_i}}{\partial\theta_j}.$$

The familiar multiple local optima problem is also present in the marginal likelihood maximization. However, practical experiences suggest that local optima are not a devastating problem especially with simple functional forms of covariance function.

Sparse Approximation

A significant problem with the Gaussian process model is associated with the computation cost of inverting the $N \times N$ Gram matrix. A number of sparse approximation methods have been proposed to overcome this high computational demand. Common to all these methods is that only a subset of the latent function values of size $M < N$ are treated exactly, and the remaining latent values are approximated with cheaper computational demand. Quiñonero-Candela and Rasmussen (2005) describe a unifying view of sparse approximation. Several existing sparse methods are shown to be an instance of it. The framework is described for regression problems; however, it should also be applicable for classification learning settings, albeit with complicacy associated with the non-Gaussian likelihood.

In this unifying treatment, an additional set of M latent variables $f_U \in \mathbb{R}^M$, called inducing variables, is introduced. These latent variables are latent function values corresponding to a set of input locations $X_U \in \mathbb{R}^{M\times d}$, called inducing inputs. The choice of inducing inputs is not restricted to only form the training or test inputs. Due to the marginalization property, introducing more latent variables will not change the distribution of the original variables. Consider (5) but as a joint distribution over latent training and test function values, $p(f_X, f_*|X, x_*)$

$$= \int p(f_X, f_*, f_U|X, X_U, x_*)\mathrm{d}f_U$$
$$= \int p(f_X, f_*|X, x_*, f_U)p(f_U)\mathrm{d}f_U, \quad (20)$$

with $p(f_U) = \mathcal{N}(0, K_{X_U,X_U})$. So far, no approximations have been introduced. Introducing the key assumption which is f_X is conditionally independent of f_* given f_U, $f_* \perp\!\!\!\perp f_X \mid f_U$, allow us to approximate (20) as

$$p(f_X, f_*|X, x_*) \approx \int p(f_*|x_*, f_U)p(f_X|X, f_U)p(f_U)\mathrm{d}f_U, \quad (21)$$

where $p(f_*|x_*, f_U)$ and $p(f_X|X, f_U)$ are again conditional multivariate Gaussians due to the GP marginalization property. Different computationally efficient algorithms in the literature correspond to different assumptions made on those two conditional distributions. Table 2 shows various sparse approximation methods with their corresponding approximated conditional distributions. For all sparse approximation methods, the com-

Gaussian Process, Table 2 Sparse approximation methods

Method	$\hat{p}(f_X \vert X, f_U)$	$\hat{p}(f_* \vert x_*, f_U)$	Ref.
SR	$\mathcal{N}(K_{X,X_U} K^{-1}_{X_U,X_U} f_U, 0)$	$\mathcal{N}(K_{x_*,X_U} K^{-1}_{X_U,X_U} f_U, 0)$	Silverman (1985)
PP	$\mathcal{N}(K_{X,X_U} K^{-1}_{X_U,X_U} f_U, 0)$	$p(f_* \vert x_*, f_U)$	Seeger et al. (2003)
SPGPs	$\mathcal{N}(K_{X,X_U} K^{-1}_{X_U,X_U} f_U, \Delta_1)$ $\Delta_1 = \text{diag}[K_{X,X} -$ $K_{X,X_U} K^{-1}_{X_U,X_U} K_{X_U,X}]$	$p(f_* \vert x_*, f_U)$	Snelson and Ghahramani (2006)
BCM	$\mathcal{N}(K_{X,X_U} K^{-1}_{X_U,X_U} f_U, \Delta_2)$	$p(f_* \vert x_*, f_U)$	Tresp (2000a)
	$\Delta_2 = \text{blockdiag}[K_{X,X} - K_{X,X_U} K^{-1}_{X_U,X_U} K_{X_U,X}]$		

SR subset of regressors, *PP* projected process, *SPGPs* sparse pseudo-input Gaussian processes, *BCM* Bayesian committee machine

putational complexity is reduced from $\mathcal{O}(N^3)$ to $\mathcal{O}(NM^2)$.

Current and Future Directions

Gaussian processes are an active area of research both within the machine learning and the Bayesian statistics community. First, there is the issue of efficient inference and learning as already discussed in the text above. Some of the recent approaches include converting Gaussian processes with stationary covariance functions to state-space models (Särkkä et al. 2013) and using stochastic variational inference (Hensman et al. 2013). Second, there is interest in adapting Gaussian processes to other learning settings. They have been used for ordinal regression (Chu and Ghahramani 2005a; Yu et al. 2006b), preference learning (Chu and Ghahramani 2005b), ranking (Guiver and Snelson 2008), mixtures of experts (Tresp 2000b), transductive learning (Schwaighofer and Tresp 2003), multitask learning (Yu et al. 2005), dimensionality reduction (Lawrence 2005), matrix factorization (Lawrence and Urtasun 2009), and reinforcement learning (Engel et al. 2005; Deisenroth and Rasmussen 2009), among other settings. They have also been extended to handle relational data (Chu et al. 2006; Yu et al. 2006a; Silva et al. 2007; Xu et al. 2009; Kersting and Xu 2009). Standard Gaussian processes only exploit the available information about attributes of instances and typically ignore any relations among the instances. Intuitively, however, we would like to use our information about one instance to help us reach conclusions about other related instances.

There is the issue of relaxing the assumption of the standard Gaussian process model that the noise on the output is uniform throughout the domain. If we assume that the noise is a smooth function of the inputs, the noise variance can be modeled using a second Gaussian process, in addition to the process governing the noise-free output values. The resulting heteroscedastic, i.e., input-dependent noise regression model, has been shown to outperform state-of-the-art methods for mobile robot localization (Plagemann et al. 2007). Heteroscedastic classification has also been explored by Hernández-Lobato et al. (2014) in the context of ▶ Learning Using Privileged Information.

Finally, Gaussian processes are also of great interest for practical applications because they naturally deal with noisy measurements, unevenly distributed observations, and fill small gaps in the data with high confidence while assigning higher predictive uncertainty in sparsely sampled areas. Two recent applications areas are Bayesian optimization for automatically tuning hyperparameters of machine-learning models (see, e.g., Snoek et al. 2012) and an automated Bayesian statistician for automating the process of statistical modeling (see, e.g., Lloyd et al. 2014).

In addition to the references embedded in the text above, we also recommend http://www.gaussian-process.org/. A highly recommended textbook is Rasmussen and Williams (2006).

Cross-References

▶ Dirichlet Process

Recommended Reading

Abrahamsen P (1992) A review of Gaussian random fields and correlation functions. Rapport 917, Norwegian Computing Center, Oslo. www.nr.no/publications/917_Rapport.ps

Bratières S, Quadrianto N, Ghahramani Z (to appear) GPstruct: Bayesian structured prediction using Gaussian processes. IEEE Trans Pattern Anal Mach Intell

Chu W, Ghahramani Z (2005a) Gaussian processes for ordinal regression. J Mach Learn Res 6:1019–1041

Chu W, Ghahramani Z (2005b) Preference learning with Gaussian processes. In: Proceedings of international conference on machine learning (ICML), Bonn

Chu W, Sindhwani V, Ghahramani Z, Keerthi S (2006) Relational learning with Gaussian processes. In: Proceedings of advances in neural information processing systems (NIPS), Vancouver

Deisenroth MP, Rasmussen CE, Peters J (2009) Gaussian process dynamic programming. Neurocomputing 72(7–9):1508–1524

Engel Y, Mannor S, Meir R (2005) Reinforcement learning with Gaussian processes. In: Proceedings of international conference on machine learning (ICML), Bonn

Guiver J, Snelson E (2008) Learning to rank with softrank and Gaussian processes. In: Proceedings of special interest group on information retrieval (SIGIR), Singapore

Hensman J, Fusi N, Lawrence N (2013) Gaussian processes for big data. In: Proceedings of uncertainty in artificial intelligence (UAI), Bellvue

Kersting K, Xu Z (2009) Learning preferences with hidden common cause relations. In: Proceedings of European conference on machine learning and principles and practice of knowledge discovery in databases (ECML PKDD), Bled

Krige DG (1951) A statistical approach to some basic mine valuation problems on the witwatersrand. J Chem Metall Mining Soc S Afr 52(6):119–139

Lawrence N (2005) Probabilistic non-linear principal component analysis with Gaussian process latent variable models. J Mach Learn Res 6:1783–1816

Lawrence N, Urtasun R (2009) Non-linear matrix factorization with Gaussian processes. In: Proceedings of international conference on machine learning (ICML), Montreal

Lloyd JR, Duvenaud D, Grosse R, Tenenbaum JB, Ghahramani Z (2014) Automatic construction and natural-language description of nonparametric regression models. In: Proceedings of association for the advancement of artificial intelligence (AAAI), Québec City

Hennig P (2013) Animating samples from Gaussian distributions. Technical report, 8. Max Planck Institute for Intelligent Systems

Hernández-Lobato D, Sharmanska V, Kersting K, Lampert C, Quadrianto N (2014) Mind the Nuisance: Gaussian process classification using privileged noise. In: Proceedings of advances in neural information processing systems (NIPS), Montreal

MacKay DJC (1992) The evidence framework applied to classification networks. Neural Comput 4(5):720–736

Matheron G (1963) Principles of geostatistics. Econ Geol 58:1246–1266

Neal R (1996) Bayesian learning in neural networks. Springer, New York

Nickisch H, Rasmussen CE (2008) Approximations for binary Gaussian process classification. J Mach Learn Res 9:2035–2078

Plagemann C, Kersting K, Pfaff P, Burgard W (2007) Gaussian beam processes: a nonparametric Bayesian measurement model for range finders. In: Proceedings of robotics: science and systems (RSS), Atlanta

Quiñonero-Candela J, Rasmussen CE (2005) A unifying view of sparse approximate Gaussian process regression. J Mach Learn Res 6:1939–1959

Rasmussen CE, Williams CKI (2006) Gaussian processes for machine learning. MIT Press, Cambridge

Särkkä S, Solin A, Hartikainen J (2013) Spatiotemporal learning via infinite-dimensional Bayesian filtering and smoothing. IEEE Signal Process Mag 30:51–61

Schwaighofer A, Tresp V (2003) Transductive and inductive methods for approximate guassian process regression. In: Proceedings of advances in neural information processing systems (NIPS), Vancouver

Seeger M, Williams CKI, Lawrence N (2003) Fast forward selection to speed up sparse Gaussian process regression. In: Proceedings of artificial intelligence and statistics (AISTATS), Key West

Silva R, Chu W, Ghahramani Z (2007) Hidden common cause relations in relational learning. In: Proceedings of advances in neural information processing systems (NIPS), Vancouver

Silverman BW (1985) Some aspects of the spline smoothing approach to non-parametric regression curve fitting. J R Stat Soc B 47(1):1–52

Snelson E, Ghahramani Z (2006) Sparse Gaussian processes using pseudo-inputs. In: Proceedings of advances in neural information processing systems (NIPS), Vancouver

Snoek J, Larochelle H, Adams RP (2012) Practical Bayesian optimization of machine learning algorithms. In: Proceedings of advances in neural information processing systems (NIPS), Lake Tahoe

Tresp V (2000a) A Bayesian committee machine. Neural Comput 12(11):2719–2741

Tresp V (2000b) Mixtures of Gaussian processes. In: Proceedings of advances in neural information processing systems (NIPS), Denver

Williams C, Barber D (1998) Bayesian classification with Gaussian processes. IEEE Trans Pattern Anal Mach Intell PAMI 20(12):1342–1351

Williams C, Rasmussen C (1996) Gaussian processes for regression. In: Proceedings of advances in neural information processing systems (NIPS), Denver

Xu Z, Kersting K, Tresp V (2009) Multi-relational learning with Gaussian processes. In: Proceedings of the international joint conference on artificial intelligence (IJCAI), Pasadena

Yu K, Tresp V, Schwaighofer A (2005) Learning Gaussian processes from multiple tasks. In: Proceedings of international conference on machine learning (ICML), Bonn

Yu K, Chu W, Yu S, Tresp V, Xu Z (2006a) Stochastic relational models for discriminative link prediction. In: Proceedings of advances in neural information processing systems (NIPS), Vancouver

Yu S, Yu K, Tresp V, Kriegel HP (2006b) Collaborative ordinal regression. In: Proceedings of international conference on machine learning (ICML), Pittsburgh

Gaussian Process Reinforcement Learning

Yaakov Engel
University of Alberta, Edmonton, AB, Canada

Definition

Gaussian process reinforcement learning generically refers to a class of ▶ reinforcement learning (RL) algorithms that use Gaussian processes (GPs) to model and learn some aspect of the problem.

Such methods may be divided roughly into two groups:

1. *Model-based methods*: Here, GPs are used to learn the transition and reward model of the ▶ Markov decision process (MDP) underlying the RL problem. The estimated MDP model is then used to compute an approximate solution to the true MDP.
2. *Model-free methods*: Here, no explicit representation of the MDP is maintained. Rather, GPs are used to learn either the MDP's value function, state–action value function, or some other quantity that may be used to solve the MDP.

This entry is concerned with the latter class of methods, as these constitute the majority of published research in this area.

Motivation and Background

▶ Reinforcement learning is a class of learning problems concerned with achieving long-term goals in unfamiliar, uncertain, and dynamic environments. Such tasks are conventionally formulated by modeling the environment as ▶ MDPs (or more generally as partially observable MDPs (▶ POMDPs)) and modeling the agent as an adaptive controller implementing an action-selection policy.

Markov Decision Processes

Let us denote by $\mathcal{P}(\mathcal{S})$ the set of probability distributions over (Borel) subsets of a set $\mathcal{S}$. A discrete time MDP is a tuple $(\mathcal{X}, \mathcal{U}, p_0, p, q, \gamma)$, where $\mathcal{X}$ and $\mathcal{U}$ are the state and action spaces, respectively; $p_0(\cdot) \in \mathcal{P}(\mathcal{X})$ is a probability density over initial states; $p(\cdot|\mathbf{x}, \mathbf{u}) \in \mathcal{P}(\mathcal{X})$ is a probability density over successor states, conditioned on the current state $\mathbf{x}$ and action $\mathbf{u}$; $q(\cdot|\mathbf{x}, \mathbf{u}) \in \mathcal{P}(\mathbb{R})$ is a probability distribution over immediate single-step rewards, conditioned on the current state and action. We denote by $R(\mathbf{x}, \mathbf{u})$ the random variable distributed according to $q(\cdot|\mathbf{x}, \mathbf{u})$. Finally, $\gamma \in [0, 1]$ is a discount factor. We assume that both p and q are stationary, that is, they do not depend explicitly on time. To maintain generality, we use $\mathbf{z}$ to denote either a state $\mathbf{x}$ or a state–action pair $(\mathbf{x}, \mathbf{u})$. This overloaded notation will allow us to present models and algorithms in a concise and unified form.

In the context of control, it is useful to make several additional definitions. A *stationary policy* $\mu(\cdot|\mathbf{x}) \in \mathcal{P}(\mathcal{U})$ is a time-independent mapping from states to action-selection probabilities. A stationary policy μ induces a Markov reward process (MRP) (Puterman 1994) via *policy-dependent* state-transition probability density, defined as (Here and in the sequel, whenever integration is performed over a finite or discrete space, the integral should be understood as a summation.)

$$p_{\mathbf{x}}^{\mu}(\mathbf{x}'|\mathbf{x}) = \int_{\mathcal{U}} d\mathbf{u}\, \mu(\mathbf{u}|\mathbf{x})\, p(\mathbf{x}'|\mathbf{u},\mathbf{x}).$$

Similarly, the policy μ may also be used to define a state–action transition probability density, defined as

$$p_{\mathbf{x},\mathbf{u}}^{\mu}(\mathbf{x}',\mathbf{u}'|\mathbf{x},\mathbf{u}) = p(\mathbf{x}'|\mathbf{u},\mathbf{x})\mu(\mathbf{u}'|\mathbf{x}').$$

Using our overloaded notational convention, we refer to either of these as $p_{\mathbf{z}}^{\mu}$. Let us denote by $\boldsymbol{\xi}(\mathbf{z})$ a path that starts at $\mathbf{z}$. Hence, for a fixed policy μ and a fixed initial state or state–action pair $\mathbf{z}_0$, the probability (density) of observing the path $\boldsymbol{\xi}(\mathbf{z}_0) = (\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_t)$ of length t is (we take $\mathbf{z}_0$ as given) $\mathbb{P}(\boldsymbol{\xi}(\mathbf{z}_0)) = \prod_{i=1}^{t} p_{\mathbf{z}}^{\mu}(\mathbf{z}_i|\mathbf{z}_{i-1})$. The *discounted return* $D^{\mu}(\boldsymbol{\xi}(\mathbf{z}))$ of a path $\boldsymbol{\xi}(\mathbf{z})$ is a random process, defined (with some abuse of notation) as

$$D^{\mu}(\mathbf{z}) = D^{\mu}(\boldsymbol{\xi}(\mathbf{z})) = \sum_{i=0}^{\infty} \gamma^{i} R(\mathbf{z}_i)|(\mathbf{z}_0 = \mathbf{z}), \tag{1}$$

where $\gamma \in [0, 1]$ is the discount factor. (When $\gamma = 1$, the policy must be proper; see Bertsekas and Tsitsiklis (1996).) The randomness in $D^{\mu}(\mathbf{z})$, for any given $\mathbf{z}$, is due both to $\boldsymbol{\xi}(\mathbf{z})$ being a random process and to the randomness, or noise, in the rewards $R(\mathbf{z}_0), R(\mathbf{z}_1), \ldots$, etc., both of which jointly constitute the *intrinsic* randomness of the MDP. Equation (1) together with the stationarity of the MDP yields the recursive formula

$$D^{\mu}(\mathbf{z}) = R(\mathbf{z}) + \gamma D^{\mu}(\mathbf{z}') \quad \text{where } \mathbf{z}' \sim p_{\mathbf{z}}^{\mu}(\cdot|\mathbf{z}). \tag{2}$$

Let us define the expectation operator $\mathbf{E}_{\xi}$ as the expectation over all possible trajectories and all possible rewards collected in them. This allows us to define the *value function* $V^{\mu}(\mathbf{z})$ as the result of applying this expectation operator to the discounted return $D^{\mu}(\mathbf{z})$, i.e.,

$$V^{\mu}(\mathbf{z}) = \mathbf{E}_{\xi} D^{\mu}(\mathbf{z}). \tag{3}$$

Applying the law of total expectation to this equation results in the MRP (fixed policy) version of the Bellman equation:

$$V^{\mu}(\mathbf{z}) = R(\mathbf{z}) + \gamma \mathbf{E}_{\mathbf{z}'|\mathbf{z}}[V^{\mu}(\mathbf{z}')]. \tag{4}$$

A policy that maximizes the expected discounted return from each state is called an optimal policy and is denoted by μ^*. In the case of stationary MDPs, there exists a *deterministic* optimal policy (this is no longer the case for POMDPs and Markov games; see Kaelbling et al. (1998) and Littman (1994)). The value function corresponding to an optimal policy is called the optimal value and is denoted by $V^* = V^{\mu^*}$. While there may exist more than one optimal policy, the optimal value function is unique (Bertsekas 1995).

Reinforcement Learning

Many of the algorithms developed for solving RL problems may be traced back to the ▸ dynamic programming *value iteration* and *policy iteration* algorithms (Bellman 1957; Bertsekas 1995; Bertsekas and Tsitsiklis 1996; Howard 1960). However, there are two major features distinguishing RL from the traditional planning framework. First, while in planning it is assumed that the environment is fully known, in RL no such assumption is made. Second, the learning process in RL is usually assumed to take place *online*, namely, concurrently with the acquirement of data by the learning agent as it interacts with its environment. These two features make solving RL problems a significantly more challenging undertaking.

An important algorithmic component of policy iteration-based RL algorithms is the estimation of either state or state–action values of a fixed policy controlling an MDP, a task known as *policy evaluation*. Sutton's TD(λ) algorithm (Sutton 1984) is an early RL algorithm that performs policy evaluation based on observed sample trajectories from the MDP, while it is being controlled by the policy being evaluated (see ► Temporal Difference Learning). In its original formulation, TD(λ) as well as many other algorithms (e.g., Watkins' ► Q-Learning 1989) employs a lookup table to store values corresponding to the MDP's states or state–action pairs. This approach clearly becomes infeasible when the size of the MDP's joint state–action space exceeds the memory capacity of modern workstations. One solution to this problem is to represent the value function using a parametric function approximation architecture and allow these algorithms to estimate the parameters of approximate value functions. Unfortunately, with few exceptions, this seemingly benign modification turns out to have ruinous consequences to the convergence properties of these algorithms. One notable exception is TD(λ), when it is used in conjunction with a function approximator $\hat{V}(\mathbf{z}) = \sum_{i=1}^{N} w_i \phi_i(\mathbf{z})$, which is linear in its tunable parameters $\mathbf{w} = (w_1, \ldots, w_N)^\top$. Under certain technical conditions, it has been shown that in this case, TD(λ) converges almost surely, and the limit of convergence is "close" (in a well-defined manner) to a projection ΠV^μ of the true value function V^μ onto the finite-dimensional space $\mathcal{H}_\phi$ of functions spanned by $\{\phi_i | i = 1, \ldots, N\}$ (Tsitsiklis and Van Roy 1996). Note that this projection is the best one may hope for, as long as one is restricted to a fixed function approximation architecture. In fact, when $\lambda = 1$, the bound of Tsitsiklis and Van Roy (1996) implies that TD(1) converges to ΠV^μ (assuming it is unique). However, as λ is reduced toward 0, the quality of TD(λ)'s solution may deteriorate significantly. If V^μ happens to belong to $\mathcal{H}_\phi$, then $V^\mu = \Pi V^\mu$ and TD(λ) converges almost surely to V^μ, for any $\lambda \in [0, 1]$.

As noted in Bertsekas and Tsitsiklis (1996), TD(λ) is a stochastic approximation algorithm (Kushner and Yin 1997). As such, to ensure convergence to a meaningful result, it relies on making small and diminishing updates to its value function estimates. Moreover, in the typical online mode of operation of TD(λ), a sample is observed, is acted upon (by updating the parameters of $\hat{V}$), and is then discarded, never to be seen again. A negative consequence of these two properties is that online TD(λ) is inherently wasteful in its use of the observed data. The least-squares TD(λ), or LSTD(λ) algorithm (Boyan 1999; Bradtke and Barto 1996), was put forward as an alternative to TD(λ) that makes better use of data, by directly solving a set of equations characterizing the fixed point of the TD(λ) updates. LSTD(λ) is amenable to a recursive implementation, at a time and memory cost of $O(N^2)$ per sample. A more fundamental shortcoming, shared by both TD(λ) and LSTD(λ), is that they do not supply the user with a measure of the accuracy of their value predictions.

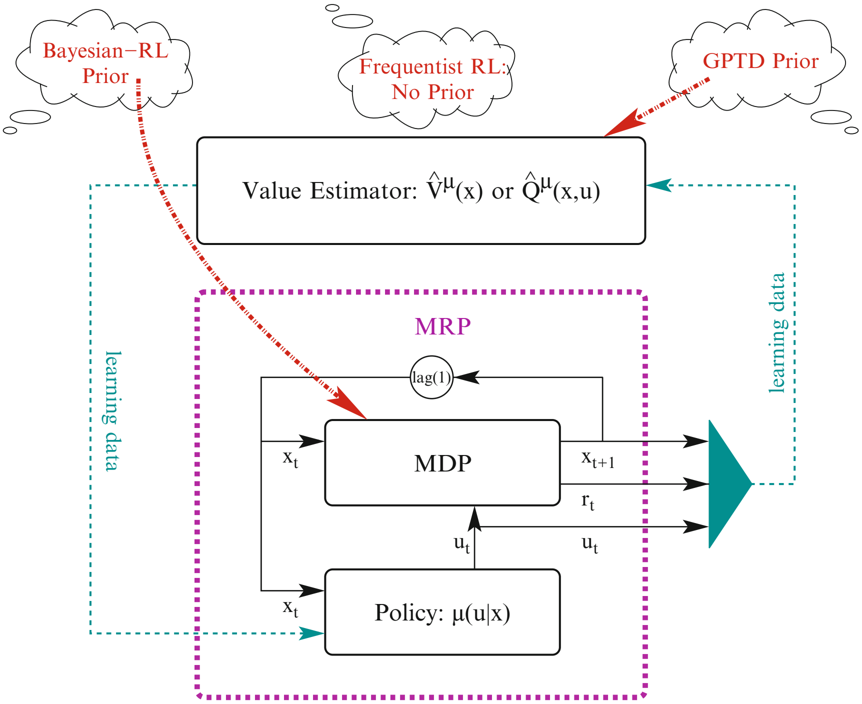
The discussion above motivates the search for:

1. Nonparametric estimators for V^μ, since these are not generally restricted to searching in any finite-dimensional hypothesis space, such as $\mathcal{H}_\phi$.
2. Estimators that make efficient use of the data.
3. Estimators that, in addition to value predictions, deliver a measure of the uncertainty in their predictions.

Structure of Learning System

We first describe the structure and operation of the basic GP temporal difference (GPTD) algorithm for policy evaluation. We then build on this algorithm to describe policy-improving algorithms, in the spirit of Howard's policy iteration (Howard 1960).

In the preceding section, we showed that the value V is the result of taking the expectation of the discounted return D with respect to the randomness in the trajectories and in the rewards collected therein. In the classic or frequentist approach, V is no longer random, since it is the true, albeit unknown value function induced by the policy μ. Adopting the Bayesian approach, we may still view the value V as a random entity by assigning it additional randomness, that is due to our *subjective uncertainty* regarding the MDP's transition model (p, q). We do not know what the true distributions p and q are,



Gaussian Process Reinforcement Learning, Fig. 1 An illustration of the frequentist as well as the two different Bayesian approaches to value function-based reinforcement learning. In the traditional Bayesian RL approach, a prior is placed on the MDP's model, whereas in our GPTD approach, the prior is placed directly on the value function. x, $\mathbf{u}$, and r denote state, action, and reward, respectively. The data required to learn value estimators typically consists of a temporal stream of state–action–reward triplets. Another stream of data is used to update the policy based on the current estimate of the value function. An MDP and a stationary policy controlling it jointly constitute an MRP. lag(1) denotes the 1-step time-lag operator

which means that we are also uncertain about the true value function. Previous attempts to apply Bayesian reasoning to RL modeled this uncertainty by placing priors over the MDP's transition and reward model (p, q) and applying Bayes' rule to update a posterior based on observed transitions. This line of work may be traced back to the pioneering works of Bellman (1956) and Howard (1960) followed by more recent contributions in the machine learning literature (Dearden et al. 1999, 1998; Duff 2002; Mannor et al. 2004; Poupart et al. 2006; Strens 2000; Wang et al. 2005). A fundamental shortcoming of this approach is that the resulting algorithms are limited to solving MDPs with finite (and typically rather small) state and action spaces, due to the need to maintain a probability distribution over the MDP's transition model. In this work, we pursue a different path – we choose to model our uncertainty about the MDP by placing a prior (and updating a posterior) directly on V. We achieve this by modeling V as a random process or, more specifically, as a Gaussian process. This mirrors the traditional classification of classical RL algorithms to either model-based or model-free (direct) methods; see Chapter 9 in Sutton and Barto (1998). Figure 1 illustrates these different approaches.

Gaussian Process Temporal Difference Learning

GPTD should be viewed as a family of *statistical generative models* (see ▸ Generative Learning) for value functions, rather than as a family of algorithms. As such, GPTD models specify the statistical relation between the unobserved value function and the observable quantities, namely, the observed trajectories and the rewards collected in them. The set of equations prescribing the GPTD model for a path $\boldsymbol{\xi} = (\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_t)$ is (Here and in the sequel, to simplify notation, we omit the superscript μ, with the understanding that quantities such as D, V, or $\boldsymbol{\xi}$ generally depend on the policy μ being evaluated.)

$$R(\mathbf{z}_i) = V(\mathbf{z}_i) - \gamma V(\mathbf{z}_{i+1}) + N(\mathbf{z}_i, \mathbf{z}_{i+1})$$
$$\text{for } i = 0, 1, \ldots, t-1.$$

$N(\mathbf{z}_i, \mathbf{z}_{i+1})$ is a zero-mean noise term that must account for the statistics of $R(\mathbf{z}_i) + \gamma V(\mathbf{z}_{i+1}) - V(\mathbf{z}_i)$. If V is a priori distributed according to a GP prior and the noise term $N(\mathbf{z}_i, \mathbf{z}_{i+1})$ is also normally distributed, then $R(\mathbf{z}_i)$ is also normally distributed and so is the posterior distribution of V conditioned on the observed rewards. To fully specify the GPTD model, we need to specify the GP prior over V in terms of prior mean and covariance as well as the covariance of the noise process N. In Engel et al. (2003), it was shown that modeling N as a white noise process is a suitable choice for MRPs with deterministic transition dynamics. In Engel et al. (2005a), a different, correlated noise model was shown to be useful for general MRPs. Let us define $R_t = (R(\mathbf{z}_0), \ldots, R(\mathbf{z}_t))$, $V_t = (V(\mathbf{z}_0), \ldots, V(\mathbf{z}_t))$, and $N_t = (N(\mathbf{z}_0, \mathbf{z}_1), \ldots, N(\mathbf{z}_{t-1}, \mathbf{z}_t))$ and also define the $t \times (t+1)$ matrix

$$\mathbf{H}_t = \begin{bmatrix} 1 & -\gamma & 0 & \ldots & 0 \\ 0 & 1 & -\gamma & & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & 1 & -\gamma \end{bmatrix}.$$

In the white noise and correlated noise GPTD models, the noise covariance matrices $\boldsymbol{\Sigma}_t = \mathrm{Cov}[N_t]$ are given, respectively, by

$$\begin{bmatrix} \sigma_R^2(\mathbf{z}_0) & 0 & \ldots & 0 \\ 0 & \sigma_R^2(\mathbf{z}_1) & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \ldots & 0 & \sigma_R^2(\mathbf{z}_{t-1}) \end{bmatrix}$$

$$\text{and} \quad \mathbf{H}_t \begin{bmatrix} \sigma_0^2 & 0 & \ldots & 0 \\ 0 & \sigma_1^2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \ldots & 0 & \sigma_t^2 \end{bmatrix} \mathbf{H}_t^\top.$$

The final component of the GPTD model remaining to be specified is the prior distribution of the GP V. This distribution is specified by prior mean and covariance functions $v_0(\mathbf{z})$ and $k(\mathbf{z}, \mathbf{z}')$, respectively.

Let us define $\boldsymbol{v}_t = (v_0(\mathbf{z}_0), \ldots, v_0(\mathbf{z}_t))^\top$. Employing ▸ Bayes' rule, the posterior distribution of $V(\mathbf{z})$ – the value function at some arbitrary query point $\mathbf{z}$ – is now given by

$$(V(\mathbf{z}) | R_{t-1} = \mathbf{r}_{t-1}) \sim \mathcal{N}\{\hat{V}_t(\mathbf{z}), P_t(\mathbf{z}, \mathbf{z}),$$

where

$$\hat{V}_t(\mathbf{z}) = v_0(\mathbf{z}) + \mathbf{k}_t(\mathbf{z})^\top \boldsymbol{\alpha}_t, \quad P_t(\mathbf{z}, \mathbf{z}')$$
$$= k(\mathbf{z}, \mathbf{z}') - \mathbf{k}_t(\mathbf{z})^\top \mathbf{C}_t \mathbf{k}_t(\mathbf{z}'),$$
$$\boldsymbol{\alpha}_t = \mathbf{H}_t^\top (\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \boldsymbol{\Sigma}_t)^{-1} (\mathbf{r}_{t-1} - \mathbf{H}_t \boldsymbol{v}_t),$$
$$\mathbf{C}_t = \mathbf{H}_t^\top (\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \boldsymbol{\Sigma}_t)^{-1} \mathbf{H}_t.$$

It is seen here that in order to compute the posterior distribution of V for arbitrary sets of query points, one only needs the vector $\boldsymbol{\alpha}_t$ and the matrix $\mathbf{C}_t$. Consequently, $\boldsymbol{\alpha}_t$ and $\mathbf{C}_t$ are sufficient statistics for the posterior of V.

Algorithms 1 and 2 provide pseudocode for recursive computations of these sufficient statistics, in the deterministic transitions and general MDP models, respectively.

It can be seen that after observing t sample transitions, both the algorithms require storage quadratic in t (due to the matrix $\mathbf{C}_t$). The updates also require time quadratic in t due to matrix-vector products involving $\mathbf{C}_t$. These properties are unsatisfying from a practical point of view, since realistic RL problems typically require large amounts of data to learn. There are two general approaches for reducing the memory and time footprints of GPTD. One approach is to define parametric counterparts of the two GPTD models described earlier and derive the corre-

Algorithm 1 Recursive nonparametric GPTD for deterministic MDPs

```
Initialize $\boldsymbol{\alpha}_0 = 0$, $\mathbf{C}_0 = 0$, $\mathcal{D}_0 = \{\mathbf{z}_0\}$
for $t = 1, 2, \ldots$
    observe $\mathbf{z}_{t-1}, r_{t-1}, \mathbf{z}_t$
    $\mathbf{h}_t = (0, \ldots, 1, -\gamma)^\top$
    $\boldsymbol{\Delta}\mathbf{k}_t = \mathbf{k}_{t-1}(\mathbf{z}_{t-1}) - \gamma \mathbf{k}_{t-1}(\mathbf{z}_t)$
    $\Delta k_{tt} = k(\mathbf{z}_{t-1}, \mathbf{z}_{t-1}) - 2\gamma k(\mathbf{z}_{t-1}, \mathbf{z}_t) + \gamma^2 k(\mathbf{z}_t, \mathbf{z}_t)$
    $\mathbf{c}_t = \mathbf{h}_t - \begin{pmatrix} \mathbf{C}_{t-1} \boldsymbol{\Delta}\mathbf{k}_t \\ 0 \end{pmatrix}$
    $d_t = r_{t-1} - \boldsymbol{\Delta}\mathbf{k}_t{}^\top \boldsymbol{\alpha}_{t-1}$
    $s_t = \sigma_{t-1}^2 + \Delta k_{tt} - \boldsymbol{\Delta}\mathbf{k}_t{}^\top \mathbf{C}_{t-1} \boldsymbol{\Delta}\mathbf{k}_t$
    $\boldsymbol{\alpha}_t = \begin{pmatrix} \boldsymbol{\alpha}_{t-1} \\ 0 \end{pmatrix} + \frac{\mathbf{c}_t}{s_t} d_t$
    $\mathbf{C}_t = \begin{bmatrix} \mathbf{C}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t} \mathbf{c}_t \mathbf{c}_t^\top$
    $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{z}_t\}$
end for
return $\boldsymbol{\alpha}_t$, $\mathbf{C}_t$, $\mathcal{D}_t$
```

Algorithm 2 Recursive nonparametric GPTD for general MDPs

```
Initialize $\boldsymbol{\alpha}_0 = \mathbf{0}$, $\mathbf{C}_0 = 0$, $\mathcal{D}_0 = \{\mathbf{z}_0\}$, $\mathbf{c}_0 = \mathbf{0}$, $d_0 = 0$, $1/s_0 = 0$
for $t = 1, 2, \ldots$
    observe $\mathbf{z}_{t-1}, r_{t-1}, \mathbf{z}_t$
    $\mathbf{h}_t = (0, \ldots, 1, -\gamma)^\top$
    $\boldsymbol{\Delta}\mathbf{k}_t = \mathbf{k}_{t-1}(\mathbf{z}_{t-1}) - \gamma \mathbf{k}_{t-1}(\mathbf{z}_t)$
    $\Delta k_{tt} = k(\mathbf{z}_{t-1}, \mathbf{z}_{t-1}) - 2\gamma k(\mathbf{z}_{t-1}, \mathbf{z}_t) + \gamma^2 k(\mathbf{z}_t, \mathbf{z}_t)$
    $\mathbf{c}_t = \frac{\gamma \sigma_{t-1}^2}{s_{t-1}} \begin{pmatrix} \mathbf{c}_{t-1} \\ 0 \end{pmatrix} + \mathbf{h}_t - \begin{pmatrix} \mathbf{C}_{t-1} \boldsymbol{\Delta}\mathbf{k}_t \\ 0 \end{pmatrix}$
    $d_t = \frac{\gamma \sigma_{t-1}^2}{s_{t-1}} d_{t-1} + r_{t-1} - \boldsymbol{\Delta}\mathbf{k}_t{}^\top \boldsymbol{\alpha}_{t-1}$
    $s_t = \sigma_{t-1}^2 + \gamma^2 \sigma_t^2 - \frac{\gamma^2 \sigma_{t-1}^4}{s_{t-1}} + \Delta k_{tt} - \boldsymbol{\Delta}\mathbf{k}_t{}^\top \mathbf{C}_{t-1} \boldsymbol{\Delta}\mathbf{k}_t + \frac{2\gamma \sigma_{t-1}^2}{s_{t-1}} \mathbf{c}_{t-1}^\top \boldsymbol{\Delta}\mathbf{k}_t$
    $\boldsymbol{\alpha}_t = \begin{pmatrix} \boldsymbol{\alpha}_{t-1} \\ 0 \end{pmatrix} + \frac{\mathbf{c}_t}{s_t} d_t$
    $\mathbf{C}_t = \begin{bmatrix} \mathbf{C}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t} \mathbf{c}_t \mathbf{c}_t^\top$
    $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{z}_t\}$
end for
return $\boldsymbol{\alpha}_t$, $\mathbf{C}_t$, $\mathcal{D}_t$
```

sponding recursive algorithms. If the number of independent parameters (i.e., the dimensionality of the hypothesis space $\mathcal{H}_\phi$) used to represent the value function is m, the memory and time costs of the algorithms become quadratic in m, rather than t. Another approach, which is based on an efficient sequential kernel sparsification method, allows us to selectively exclude terms from $\mathcal{D}_t$, while controlling the error incurred as a result. Here again (bounds on m in this case may be derived using arguments based on the finiteness of packing numbers of the hypothesis space; see Engel (2005) for details), if the size of $\mathcal{D}_t$ saturates at m, the memory and time costs of the resulting algorithms are quadratic in m. For the complete derivations, as well as detailed pseudocode of the corresponding algorithms, we refer the reader to Engel (2005).

Theory

In this section, we derive the two GPTD models mentioned above, explicitly stating the assumptions underlying each model.

MRPs with Deterministic Transitions

In the deterministic case, the Bellman equation (4) degenerates into

$$\bar{R}(\mathbf{z}) = V(\mathbf{z}) - \gamma V(\mathbf{z}'), \qquad (5)$$

where $\mathbf{z}'$ is the state or state–action pair succeeding $\mathbf{z}$, under the deterministic policy μ. We also assume that the noise in the rewards is independent and Gaussian, but not necessarily identically distributed. We denote the reward variance by $\sigma_R^2(\mathbf{z}) = \mathrm{Var}[R(\mathbf{z})]$. Formally, this means that the reward $R(\mathbf{z})$, at some $\mathbf{z}$, satisfies $R(\mathbf{z}) = \bar{R}(\mathbf{z}) + N(\mathbf{z})$ where $\bar{R}(\mathbf{z})$ is the mean reward for that state. Assume we have a sequence of rewards sampled along a sampled path $\boldsymbol{\xi}$. Then, at the ith time step, we have $R(\mathbf{z}_i) = \bar{R}(\mathbf{z}_i) + N(\mathbf{z}_i)$. Using the random vectors R_t, V_t, and N_t defined earlier, we have $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_t)$, where

$$\boldsymbol{\Sigma}_t = \mathrm{diag}(\sigma_R^2(\mathbf{z}_0), \ldots, \sigma_R^2(\mathbf{z}_{t-1})), \qquad (6)$$

and diag$(\cdot)$ denotes a diagonal matrix whose diagonal elements are the components of the argument vector. Writing the Bellman equations (5) for the points belonging to the sample path and substituting $R(\mathbf{z}_i) = \bar{R}(\mathbf{z}_i) + N(\mathbf{z}_i)$, we obtain the following set of t equations

$$R(\mathbf{z}_i) = V(\mathbf{z}_i) - \gamma V(\mathbf{z}_{i+1}) + N(\mathbf{z}_i),$$
$$i = 0, 1, \ldots, t-1.$$

This set of linear equations may be concisely written as

$$R_{t-1} = \mathbf{H}_t V_t + N_t. \tag{7}$$

General MRPs

Let us consider a decomposition of the discounted return D into its mean V and a zero-mean residual ΔV:

$$D(\mathbf{z}) = \mathbf{E}_\xi D(\mathbf{z}) + (D(\mathbf{z}) \\ -\mathbf{E}_\xi D(\mathbf{z})\big) \overset{\text{def}}{=} V(\mathbf{z}) + \Delta V(\mathbf{z}). \tag{8}$$

This decomposition is useful, since it separates the two sources of uncertainty inherent in the discounted return process D: For a known MDP model, V is a (deterministic) function, and the randomness in D is fully attributed to the intrinsic randomness in the trajectories generated by the MDP and policy pair, modeled by ΔV. On the other hand, in an MDP in which both transitions and rewards are deterministic but otherwise unknown, ΔV is deterministic (identically zero), and the randomness in D is due solely to the extrinsic Bayesian uncertainty, modeled by the random process V.

Substituting (8) into (2) and Rearranging, we get

$$R(\mathbf{z}) = V(\mathbf{z}) - \gamma V(\mathbf{z}') + N(\mathbf{z}, \mathbf{z}'),$$

where $\mathbf{z}' \sim p^{\mu}(\cdot|\mathbf{z})$ and

$$N(\mathbf{z}, \mathbf{z}') \overset{\text{def}}{=} \Delta V(\mathbf{z}) - \gamma \Delta V(\mathbf{z}'). \tag{9}$$

As before, we are provided with a sample path $\boldsymbol{\xi}$, and we may write the model Eqs. (9) for these samples, resulting in the following set of t equations:

$$R(\mathbf{z}_i) = V(\mathbf{z}_i) - \gamma V(\mathbf{z}_{i+1}) + N(\mathbf{z}_i, \mathbf{z}_{i+1}) \\ \text{for } i = 0, \dots, t-1.$$

Using our standard definitions for R_t, V_t, $\mathbf{H}_t$, and with $N_t = (N(\mathbf{z}_0, \mathbf{z}_1), \dots, N(\mathbf{z}_{t-1}, \mathbf{z}_t))^\top$, we again have

$$R_{t-1} = \mathbf{H}_t V_t + N_t. \tag{10}$$

In order to fully define a complete probabilistic generative model, we also need to specify the distribution of the noise process N_t. We model the residuals $\Delta V_t = (\Delta V(\mathbf{z}_0), \dots, \Delta V(\mathbf{z}_t))^\top$ as random Gaussian noise. (This may not be a correct assumption in general; however, in the absence of any prior information concerning the distribution of the residuals, it is the *simplest* assumption we can make, since the Gaussian distribution possesses the highest entropy among all distributions with the same covariance. It is also possible to relax the Gaussianity requirement on both the prior and the noise. The resulting estimator may then be shown to be the *linear minimum mean-squared error* estimator for the value.) In particular, this means that the distribution of the vector ΔV_t is completely specified by its mean and covariance. Another assumption we make is that each of the residuals $\Delta V(\mathbf{z}_i)$ is independently distributed. Denoting $\sigma_i^2 = \mathrm{Var}[D(\mathbf{z}_i)]$, the distribution of ΔV_t is given by

$$\Delta V_t \sim \mathcal{N}(\mathbf{0}, \mathrm{diag}(\boldsymbol{\sigma}_t)),$$

where $\boldsymbol{\sigma}_t = \left(\sigma_0^2, \sigma_1^2, \dots, \sigma_t^2\right)^\top$. Since $N_t = \mathbf{H}_t \Delta V_t$, we have $N_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_t)$ with

$$\boldsymbol{\Sigma}_t = \mathbf{H}_t \mathrm{diag}(\boldsymbol{\sigma}_t) \mathbf{H}_t^\top \\ = \begin{bmatrix} \sigma_0^2 + \gamma^2\sigma_1^2 & -\gamma\sigma_1^2 & 0 & \dots & 0 & 0 \\ -\gamma\sigma_1^2 & \sigma_1^2 + \gamma^2\sigma_2^2 & -\gamma\sigma_2^2 & 0 & \dots & 0 \\ 0 & -\gamma\sigma_2^2 & \sigma_2^2 + \gamma^2\sigma_3^2 & \ddots & & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \vdots & & \ddots & \ddots & -\gamma\sigma_{t-1}^2 \\ 0 & 0 & \dots & 0 & -\gamma\sigma_{t-1}^2 & \sigma_{t-1}^2 + \gamma^2\sigma_t^2 \end{bmatrix}. \tag{11}$$

Applications

Any RL algorithm that requires policy evaluation as an algorithmic component can potentially use a GPTD algorithm for this task. In particular, this is true of algorithms based on Howard's policy iteration. In Engel et al. (2005a) and Engel (2005), it is shown how GPTD may be used to construct a SARSA-type algorithm (Rummery and Niranjan 1994; Sutton and Barto 1998), called GPSARSA. In Engel et al. (2005b), GPSARSA was used to learn control policies for a simulated Octopus arm. In Ghavamzadeh and Engel (2007), GPTD was used within a Bayesian actor–critic learning algorithm.

Future Directions

By virtue of the posterior covariance, GPTD algorithms compute a confidence measure (or, more precisely, Bayesian *credible intervals*) for their value estimates. So far, little use has been made of this additional information. Several potential uses of the posterior covariance may be envisaged:

1. It may be used to construct stopping rules for value estimation.
2. It may be used to guide exploration.
3. In the context of Bayesian actor–critic algorithms (Ghavamzadeh and Engel 2007), it may be used to control the size and direction of policy updates.

Further Reading

Yaakov Engel's doctoral thesis (Engel 2005) is currently the most complete reference to GPTD methods. Two conference papers (Engel et al. 2003, 2005a) provide a more concise view. The first of these introduces the GPTD model for deterministic MRPs, while the second introduces the general MDP model, as well as the GPSARSA algorithm. A forthcoming journal article will subsume these two papers and include some additional results, concerning the connection between GPTD and the popular TD(λ) and LSTD(λ) algorithms.

Recommended Reading

Bellman RE (1956) A problem in the sequential design of experiments. Sankhya 16:221–229

Bellman RE (1957) Dynamic programming. Princeton University Press, Princeton

Bertsekas DP (1995) Dynamic programming and optimal control. Athena Scientific, Belmont

Bertsekas DP, Tsitsiklis JN (1996) Neuro-dynamic programming. Athena Scientific, Belmont

Boyan JA (1999) Least-squares temporal difference learning.
In: Proceedings of the 16th international conference on machine learning, Bled. Morgan Kaufmann, San Francisco, pp 49–56

Bradtke SJ, Barto AG (1996) Linear least-squares algorithms for temporal difference learning. Mach Learn 22:33–57

Dearden R, Friedman N, Andre D (1999) Model based Bayesian exploration. In: Proceedings of the fifteenth conference on uncertainty in artificial intelligence, Stockholm. Morgan Kaufmann, San Francisco, pp 150–159

Dearden R, Friedman N, Russell S (1998) Bayesian Q-learning. In: Proceedings of the fifteenth national conference on artificial intelligence, Madison. AAAI, Menlo Park, pp 761–768

Duff M (2002) Optimal learning: computational procedures for Bayes-adaptive Markov decision processes. PhD thesis, University of Massachusetts, Amherst

Engel Y (2005) Algorithms and representations for reinforcement learning. PhD thesis, The Hebrew University of Jerusalem

Engel Y, Mannor S, Meir R (2003) Bayes meets Bellman: the Gaussian process approach to temporal difference learning. In: Proceedings of the 20th international conference on machine learning, Washington, DC. Morgan Kaufmann, San Francisco

Engel Y, Mannor S, Meir R (2005) Reinforcement learning with Gaussian processes. In: Proceedings of the 22nd international conference on machine learning, Bonn

Engel Y, Szabo P, Volkinshtein D (2005) Learning to control an Octopus arm with Gaussian process temporal difference methods. Technical report, Technion Institute of Technology. www.cs.ualberta.ca/~yaki/reports/octopus.pdf

Ghavamzadeh M, Engel Y (2007) Bayesian actor-critic algorithms. In: Ghahramani Z (ed) 24th international conference on machine learning, Corvalis. Omnipress, Corvallis

Howard R (1960) Dynamic programming and Markov processes. MIT, Cambridge

Kaelbling LP, Littman ML, Cassandra AR (1998) Planning and acting in partially observable stochastic domains. Artif Intell 101:99–134
Kushner HJ, Yin CJ (1997) Stochastic approximation algorithms and applications. Springer, Berlin
Littman ML (1994) Markov games as a framework for multi-agent reinforcement learning. In: Proceedings of the 11th international conference on machine learning (ICML-94), New Brunswick. Morgan Kaufmann, New Brunswick, pp 157–163
Mannor S, Simester D, Sun P, Tsitsiklis JN (2004) Bias and variance in value function estimation. In: Proceedings of the 21st international conference on machine learning, Banff
Poupart P, Vlassis NA, Hoey J, Regan K (2006) An analytic solution to discrete Bayesian reinforcement learning. In: Proceedings of the twenty-third international conference on machine learning, Pittsburgh, pp 697–704
Puterman ML (1994) Markov decision processes: discrete stochastic dynamic programming. Wiley, New York
Rummery G, Niranjan M (1994) On-line Q-learning using connectionist systems. Technical report CUED/F-INFENG/TR 166, Cambridge University Engineering Department
Strens M (2000) A Bayesian framework for reinforcement learning. In: Proceedings of the 17th international conference on machine learning, Stanford. Morgan Kaufmann, San Francisco, pp 943–950
Sutton RS (1984) Temporal credit assignment in reinforcement learning. PhD thesis, University of Massachusetts, Amherst
Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT, Cambridge
Tsitsiklis JN, Van Roy B (1996) An analysis of temporal-difference learning with function approximation. Technical report LIDS-P-2322. MIT, Cambridge
Wang T, Lizotte D, Bowling M, Schuurmans D (2005) Bayesian sparse sampling for on-line reward optimization. In: Proceedings of the 22nd international conference on machine learning, Bonn. ACM, New York, pp 956–963
Watkins CJCH (1989) Learning from delayed rewards. PhD thesis, King's College, Cambridge

Gaussian Processes

▶ Bayesian Nonparametric Models

Generality and Logic

▶ Logic of Generality

Generalization

Claude Sammut
The University of New South Wales, Sydney,
NSW, Australia

A hypothesis, h, is a predicate that maps an instance to *true* or *false*. That is, if $h(x)$ is true, then x is hypothesized to belong to the concept being learned, the *target*. Hypothesis, h_1, is more general than or equal to h_2, if h_1 covers at least as many examples as h_2 (Mitchell, 1997). That is, $h_1 \geq h_2$ if and only if

$$(\forall x)[h_1(x) \rightarrow h_2(x)]$$

A hypothesis, h_1, is strictly more general than h_2, if $h_1 \geq h_2$ and $h_2 \not\leq h_1$.

Note that the *more general than* ordering is strongly related to *subsumption*.

Cross-References

▶ Classification
▶ Induction
▶ Learning as Search
▶ Logic of Generality
▶ Specialization
▶ Subsumption

Recommended Reading

Mitchell TM (1997) Machine learning. McGraw-Hill, New York

Generalization Bounds

Mark Reid
The Australian National University, Canberra,
ACT, Australia

Synonyms

Inequalities; Sample complexity

Definition

In the theory of statistical machine learning, a generalization bound – or, more precisely, a generalization error bound – is a statement about the predictive performance of a learning algorithm or class of algorithms. Here, a learning algorithm is viewed as a procedure that takes some finite training sample of labeled instances as input and returns a hypothesis regarding the labels of all instances, including those which may not have appeared in the training sample. Assuming labeled instances are drawn from some fixed distribution, the quality of a hypothesis can be measured in terms of its risk – its incompatibility with the distribution. The performance of a learning algorithm can then be expressed in terms of the expected risk of its hypotheses given randomly generated training samples.

Under these assumptions, a generalization bound is a theorem, which holds for any distribution and states that, with high probability, applying the learning algorithm to a randomly drawn sample will result in a hypothesis with risk no greater than some value. This bounding value typically depends on the size of the training sample, an empirical assessment of the risk of the hypothesis on the training sample as well as the "richness" or "capacity" of the class of predictors that can be output by the learning algorithm.

Motivation and Background

Suppose we have built an e-mail classifier and then collected a random sample of e-mail labeled as "spam" or "not spam" to test it on. We notice that the classifier incorrectly labels 5 % of the sample. What can be said about the accuracy of this classifier when it is applied to new, previously unseen e-mail? If we make the reasonable assumption that the mistakes made on future e-mails are independent of mistakes made on the sample, basic results from statistics tell us that the classifier's true error rate will also be around 5 %.

Now suppose that instead of building a classifier by hand we use a learning algorithm to *infer* one from the sample. What can be said about the future error rate of the inferred classifier if it also misclassifies 5 % of the training sample? In general, the answer is "nothing" since we can no longer assume future mistakes are independent of those made on the training sample. As an extreme case, consider a learning algorithm that outputs a classifier that just "memorizes" the training sample – predicts labels for e-mail in the sample according to what appears in the sample – and predicts randomly otherwise. Such a classifier will have a 0 % error rate on the sample, however, if most future e-mail does not appear in the training sample the classifier will have a true error rate around 50 %.

To avoid the problem of memorizing or overfitting the training data it is necessary to restrict the "flexibility" of the hypotheses a learning algorithm can output. Doing so forces predictions made off the training set to be related to those made on the training set so that some form of generalization takes place. However, doing this can limit the ability of the learning algorithm to output a hypothesis with small risk. Thus, there is a classic and trade-off: the bias being the limits placed on how flexible the hypotheses can be versus the variance between the training and the true error rates (see bias variance decomposition).

By quantifying the notion of hypothesis flexibility in various ways, generalization bounds provide inequalities that show how the flexibility and empirical error rate can be traded off to control the true error rate. Importantly, these statements are typically probabilistic but distribution-independent – they hold for nearly all sets of training data drawn from a fixed but unknown distribution. When such a bound holds for a learning algorithm it means that, unless the choice of training sample was very unlucky, we can be confident that some form of generalization will take place. The first results of this kind were established by Vapnik and Chervonenkis (1971) about 40 years ago and the measure of hypothesis flexibility they introduced – the ▸ VC dimension (see below) – now bears their initials. A similar style of results were obtained independently by Valiant in 1984 in the Probably Approximately Correct, or ▸ PAC learning framework (Valiant 1984). These two lines of work were drawn together by Blumer et al. (1989) and now form the basis of what is known today as statistical learning theory.

Details

For simplicity, we restrict our attention to generalization bounds for binary ▸ classification problems such as the spam classification example above. In this setting *instances* (e.g., e-mail) from a set $\mathcal{X}$ are associated with *labels* from a set $\mathcal{Y} = \{-1, 1\}$ (e.g., indicating not spam/spam) and an *example* $z = (x, y)$ is a labeled instance from $\mathcal{Z} := \mathcal{X} \times \mathcal{Y}$. The association of instances to labels is assumed to be governed by some unknown distribution P over $\mathcal{Z}$.

A *hypothesis* h is a function that assigns labels $h(x) \in \mathcal{Y}$ to instances. The quality of a hypothesis is assessed via a *loss* function $\ell : \mathcal{Y} \times \mathcal{Y} \to [0, \infty$, which assigns penalty $\ell(y, y')$ when h predicts the label $y' = h(x)$ for the example (x, y). For convenience, we will often combine the loss and hypothesis evaluation on an example $z = (x, y)$ by defining $\ell_h(z) = \ell(y, h(x))$. When examples are sampled from P the expected penalty, or *risk*

$$L_p(h) := \mathbb{E}_P[\ell_h(z)]$$

can be interpreted as a measure of how well h models the distribution P. A loss that is prevalent in classification is the *0–1 loss* $\ell^{0-1}(y, y') = [y \neq y']$ where $[p]$ is the indicator function for the predicate p. This loss simply assigns a penalty of 1 for an incorrect prediction and 0 otherwise. The associated 0–1 risk for h is the probability the prediction $h(x)$ disagrees with a randomly drawn sample (x, y) from P. Unless stated otherwise, the bounds discussed below are for the 0–1 loss only but, with care, can usually be made to hold with more general losses also.

Once a loss is specified, the goal of a learning algorithm is to produce a low-risk hypothesis based on a finite number of examples. Formally, a *learning algorithm* $\mathcal{A}$ is a procedure that takes a *training sample* $\mathbf{z} = (z_1, \ldots, z_n) \in \mathcal{Z}^n$ as input and returns a hypothesis $h = \mathcal{A}(\mathbf{z})$ with an associated *empirical risk*

$$\hat{L}_{\mathbf{z}}(h) := \frac{1}{n} \sum_{i=1}^{n} \ell_h(z_i).$$

In order to relate the empirical and true risks, a common assumption made in statistical learning theory is that the examples are drawn independently from P. In this case, a sample $\mathbf{z} = (z_1, \ldots, z_n)$ is a random variable from the product distribution P^n over $\mathcal{Z}^n$. Since the sample can be of arbitrary but finite size a learning algorithm can be viewed as a function $\mathcal{A} : \bigcup_{n=1}^{\infty} \mathcal{Z}^n \to \mathcal{H}$ where $\mathcal{H}$ is the algorithm's ▸ hypothesis space.

A generalization bound typically comprises several quantities: an empirical estimate of a hypothesis's performance $\hat{L}_{\mathbf{z}}(h)$; the actual (and unknown) risk of the hypothesis $L_P(h)$; a confidence term $\delta \in [0, 1]$; and some measure of the flexibility or *complexity* C of the hypotheses that can be output by learning algorithm. The majority of the bounds found in the literature fit the following template.

▸ *A generic generalization bound*: Let $\mathcal{A}$ be a learning algorithm, P some unknown distribution over $\mathcal{X} \times \mathcal{Y}$, and $\delta > 0$. Then, with probability at least $1 - \delta$ over randomly drawn samples $\mathbf{z}$ from P^n, the hypothesis $h = \mathcal{A}(\mathbf{z})$ has risk $L_P(h)$ no greater than $\hat{L}_{\mathbf{z}}(h) + \epsilon(\delta, C)$.

Of course, there are many variations, refinements, and improvements of the bounds presented below and not all fit this template. The bounds discussed below are only intended to provide a survey of some of the key ideas and main results.

Basic bounds: The penalties $\ell_h(z_i) := \ell(y_i, h(x_i))$ made by a fixed hypothesis h on a sample $\mathbf{z} = (z_1, \ldots, z_n)$ drawn from P^n are independent random variables. The law of large numbers guarantees (under some mild conditions) that their mean $\hat{L}_{\mathbf{z}}(h) = \frac{1}{n}\sum_{i=1}^{n} \ell_h(z_i)$ converges to the true risk $L_P(h) = \mathbb{E}_P[\ell_h(z)]$ for h as the sample size increases and several inequalities from probability theory can be used to quantify this convergence. A key result is ▸ McDiarmid's inequality, which can be used to bound the deviation of a function of independent random variables from its mean. Since the 0–1 loss takes values in [0, 1], applying this result to the random variables $\ell_h(Z_i)$ gives

$$P^n(L_P(h) > \hat{L}_{\mathbf{z}}(h) + \varepsilon) \le \exp(-2n\varepsilon^2). \quad (1)$$

We can invert this and obtain an upper bound for the true risk that will hold on a given proportion of samples. That is, if we want $L_P(h) > \hat{L}_{\mathbf{z}}(h) + \epsilon$ to hold on at least $1 - \delta$ of the time on randomly drawn samples we can solve $\delta = \exp(-2n\varepsilon^2)$ for ε and obtain $\varepsilon = \sqrt{\frac{\ln\frac{1}{\delta}}{2n}}$ so that

$$P^n\left(L_P(h) \le \hat{L}_{\mathbf{z}}(h) + \sqrt{\frac{\ln\frac{1}{\delta}}{2n}}\right) \ge 1 - \delta. \quad (2)$$

This simple bound lays the foundation for many of the subsequent bounds discussed below and is the reason for the ubiquity of the $\sqrt{\frac{\ln\frac{1}{\delta}}{n}}$-like terms.

A crucial observation to make about the above bound is that while it holds for any hypothesis h it does *not* hold for all $h \in \mathcal{H}$ *simultaneously*. That is, the samples for which the bounds hold for h_1 may be completely different to those which make the bound hold for h_2. Since a generalization bound must hold for all possible hypotheses output by a learning algorithm we need to extend the above analysis by exploiting additional properties of the hypothesis space $\mathcal{H}$.

In the simple case when there are only finitely many hypothesis, we use the *union bound.* This states that for any distribution P and any finite or countably infinite sequence of events A_1, $A_2 \ldots$ we have $P(\bigcup_i A_i) \le \sum_i P(A_i)$. For $\mathcal{H} = \{h_1, \ldots, h_m\}$ we consider the events $Z_h = \{\mathbf{z} \in \mathcal{Z}^n : L_P(h) > \hat{L}_{\mathbf{z}}(h) + \epsilon\}$ when samples of size n give empirical risks for h that are least ε smaller than its true risk. Using the union bound and (1) on these events gives

$$P^n\left(\bigcup_{h \in \mathcal{H}} Z_h(n, \varepsilon)\right) \le \sum_{i=1}^{m} P^n(Z_h(n, \varepsilon))$$
$$= m \cdot \exp(-2n\varepsilon^2).$$

This is a bound on the probability of drawing a training sample from P^n such that *every* hypothesis has a true risk that is ε larger than its empirical risk. Inverting this inequality by setting $\delta = m\exp(-2n\varepsilon^2)$ yields the following bound.

Finite class bound: Suppose $\mathcal{A}$ has finite hypothesis class $\mathcal{H} = \{h_1, \ldots, h_m\}$. Then with probability at least $1 - \delta$ over draws of $\mathbf{z}$ from P^n the hypothesis $h = \mathcal{A}(\mathbf{z})$ satisfies

$$L_P(h) \le \hat{L}_{\mathbf{z}}(h) + \sqrt{\frac{\ln|\mathcal{H}| + \ln\frac{1}{\delta}}{2n}}. \quad (3)$$

It is instructive to compare this to the single hypothesis bound in (2) and note the bound is weakened by the additional term $|\mathcal{H}|$.

Since the union bound also holds for countable sets of events this style of bound can be extended from finite hypothesis classes to countable ones. To do this requires a slight modification of the above argument and the introduction of a distribution π over a countable hypothesis space $\mathcal{H} = \{h_1, h_2, \ldots\}$, which is chosen before any samples are seen. This distribution can be interpreted as a prior belief or preference over the hypotheses in $\mathcal{H}$. Letting $\delta(h) = \delta \cdot \pi(h)$ in the bound (2) implies that for each $\mathcal{H}$ we have

$$P^n\left(L_P(h) < \hat{L}_{\mathbf{z}}(h) + \sqrt{\frac{\ln\frac{1}{\delta \cdot \pi(h)}}{2n}}\right) < \delta \cdot \pi(h).$$

Thus, applying the countable union bound to the union of these events over all of $\mathcal{H}$, and noting that $\sum_{h \in \mathcal{H}} \delta.\ \pi(h) = \delta$ since π is a distribution over $\mathcal{H}$, gives use the following bound:

Countable class bound: Suppose μ is a probability distribution over a finite or countably infinite hypothesis space $\mathcal{H}$. Then with probability at least $1 - \delta$ over draws of $\mathbf{z}$ from P^n the following bound holds for all $h \in \mathcal{H}$

$$L_P(h) \le \hat{L}_{\mathbf{z}}(h) + \sqrt{\frac{\ln\frac{1}{\pi(h)} + \ln\frac{1}{\delta}}{2n}}. \quad (4)$$

Although the finite and countable class bounds are proved using very similar techniques (indeed, the former can be derived from the latter by choosing $\pi(h) = \frac{1}{|\mathcal{H}|}$), they differ in the type of penalty they introduce for simultaneously bounding all the hypotheses in $\mathcal{H}$. In (3), the penalty $\ln|\mathcal{H}|$ is purely a function of the size of the class whereas in (4) the penalty $\ln \frac{1}{\pi(h)}$ varies with h. These two different styles of bound can be seen as templates for the two main classes of bounds discussed below: the hypothesis-independent bounds of the next section and the hypothesis-dependent bounds in the section on PAC-Bayesian bounds. The main conceptual leap from here is the extension of the arguments above to non-countable hypothesis classes.

> *Class complexity bounds*: A key result in extending the notion of size or complexity in the above bounds to more general classes of hypotheses is the *symmetrization lemma*. Intuitively, it is based on the observation that if the empirical risks for different samples are frequently near the true risk then they will also be near each other. Formally, it states that for any $\varepsilon > 0$ such that $n\varepsilon^2 \geq 2$ we have
>
> $$P^n\left(\sup_{h\in\mathcal{H}} |L_P(h) - \hat{L}_z(h)| > \varepsilon\right)$$
>
> $$\leq 2P^{2n}\left(\sup_{h\in\mathcal{H}} |\hat{L}_{z'}(h) - \hat{L}_z(h)| > \frac{\varepsilon}{2}\right). \quad (5)$$

Thus, to obtain a bound on the difference between empirical and true risk it suffices to bound the difference in empirical risks on two independent samples $\mathbf{z}$ and $\mathbf{z}'$, both drawn from P^n. This is useful since the maximum difference $\sup_{h\in\mathcal{H}} |\hat{L}_{z'}(h) - \hat{L}_z(h)|$ is much easier to handle than the difference involving $L_P(h)$ as the former term only evaluates losses on the points in $\mathbf{z}$ and $\mathbf{z}'$ while the latter takes into account the entire space $\mathcal{Z}$.

To study these restricted evaluations, we define the restriction of a function class $\mathcal{F}$ to the sample $\mathbf{z}$ by $\mathcal{F}_z = \{(f(z_1), \ldots, f(z_n)) : f \in \mathcal{F}\}$. Since the empirical risk $\hat{L}_z(h) = \frac{1}{n}\Sigma_{i=1}^n \ell_h(z_i)$ only depends on the values of the loss functions ℓ_h on samples from $\mathbf{z}$ we define the *loss class* $\mathbb{L} = \ell_{\mathcal{H}} = \{\ell_h : h \in \mathcal{H}\}$ and consider its restriction $\mathbb{L}_{\mathbf{z}}$ as well as the restriction $\mathcal{H}_z$ of the hypothesis class it is built upon. As we will see, the measures of complexity of these two classes are closely related.

One such complexity measure is arrived at by examining the size of a restricted function class $\mathcal{F}_z$ as the size of the sample $\mathbf{z}$ increases. The *growth function* or ▶ shattering coefficient for the function class $\mathcal{F}$ is defined as the maximum number of distinct values the vectors in $\mathcal{F}_z$ can take given a sample of size n : $S_n(\mathcal{F}) = \sup_{z\in\mathcal{Z}^n} |\mathcal{F}_z|$. In the case of binary classification with a 0–1 loss, it is not hard to see that the growth functions for both $\mathbb{L}$ and $\mathcal{F}$ are equal, that is, $S_n(\mathrm{L}) = S_n(\mathcal{H})$, and so they can be used interchangeably. Applying a union bound argument to (1) as in the previous bounds guarantees that $P^n(\sup_{h\in\mathcal{H}} |L_P(h) - \hat{L}_z(h)| > \varepsilon) \leq 2S_n(\mathcal{H})\exp(-n\varepsilon^2/8)$ and by inversion we obtain the following generalization bound for arbitrary hypothesis classes $\mathcal{H}$:

> *Growth function bound*: For all $\delta > 0$, a draw of $\mathbf{z}$ from P^n will, with probability at least $1 - \delta$, satisfy for all $h \in \mathcal{H}$
>
> $$L_P(h) \leq \hat{L}_{\mathbf{z}}(h) + 2\sqrt{\frac{2\ln S_n(\mathcal{H}) + 2\ln\frac{2}{\delta}}{n}}. \quad (6)$$

One conclusion that can be immediately drawn from this bound is that the shattering coefficient must grow sub-exponentially for the bound to provide any meaningful guarantee. If the class $\mathcal{H}$ is so rich that hypotheses from it can fit all 2^n possible label combinations – if $S_n(\mathcal{H}) = 2^n$ for all n – then the term $\sqrt{2\ln S_n(\mathcal{H})/n} > 1$ and so (6) just states $L_P(h) \leq 1$. Therefore, to get nontrivial bounds from (6) there needs to exist some value d for which $S_n(\mathcal{H}) < 2^n$ whenever $n > d$.

VC dimension: This desired property of the growth function is exactly what is captured by the ▶ VC dimension $VC(\mathcal{H})$ of a hypothesis class $\mathcal{H}$. Formally, it is defined as $VC(\mathcal{H}) = \max\{n \in \mathbb{N} : S_n(\mathcal{H}) = 2^n\}$ and is infinite if no finite maximum exists. Whether or not the VC dimension is finite plays a central role in the consistency of empirical risk minimization techniques. Indeed, it is possible to show that using ERM on a hypothesis class $\mathcal{H}$ is consistent if and only if

$VC(\mathcal{H}) < \infty$. This is partly due to *Sauer's lemma*, which shows that when a hypothesis class $\mathcal{H}$ has finite VC dimension $VC(\mathcal{H}) = d_{\mathcal{H}} < \infty$ its growth function is eventually polynomial in the sample size. Specifically, for all $n \geq d_{\mathcal{H}}$ the growth function satisfies $S_n(\mathcal{H}) \leq \left(\frac{en}{d_{\mathcal{H}}}\right)^{d_{\mathcal{H}}}$. By substituting this result into the Growth Function Bound (6) we obtain the following bound, which shows how the VC dimension plays a role that is analogous to the size a hypothesis class in the finite case.

▸ *VC dimension bound*: Suppose $\mathcal{A}$ has hypothesis class $\mathcal{H}$ with finite VC dimension $d_{\mathcal{H}}$. Then with probability at least $1 - \delta$ over draws of $\mathbf{z}$ from P^n the hypothesis $h = \mathcal{A}(\mathbf{z})$ satisfies

$$L_P(h) \leq \hat{L}_z(h) + 2\sqrt{\frac{2d_{\mathcal{H}} \ln\left(\frac{2en}{d_{\mathcal{H}}}\right) + 2\ln\frac{2}{\delta}}{n}}. \tag{7}$$

There are many other bounds in the literature that are based on the VC dimension. See the Recommended Reading for pointers to these.

Rademacher averages: Rademacher averages are a second kind of measure of complexity for uncountable function classes and can be used to derive more refined bounds than those above. These averages arise naturally by treating as a random variable the sample-dependent quantity $M_{\mathcal{F}}(\mathbf{z}) = \sup_{f \in \mathcal{F}}[\mathbb{E}_P[f] - \mathbb{E}_{\mathbf{z}}[f]]$. This is just the largest difference taken over all $f \in \mathcal{F}$ between its true mean $\mathbb{E}_P[f]$ and its empirical mean $\mathbb{E}_{\mathbf{z}}[f] := \frac{1}{|\mathbf{z}|}\sum_{i=1}^{|\mathbf{z}|} f(z_i)$. For a loss class $\mathbb{L} = \ell_{\mathcal{H}}$ a bound on this maximum difference – $M_{\mathbb{L}}(\mathbf{z}) \leq B$ – immediately gives a generalization bound of the form $L_P(h) \leq \hat{L}_{\mathbf{z}}(h) + B$. Since $M_{\mathcal{F}}(\mathbf{z})$ is a random variable, McDiarmid's inequality can be used to bound its value in terms of its expected value plus the usual $\sqrt{\frac{\ln\frac{1}{\delta}}{2n}}$ term. Applying symmetrization it can then be shown that this expected value satisfies

$$\mathbb{E}_{P^n}[M_{\mathcal{F}}(\mathbf{z})] \leq \mathbb{E}\left[\sup_{f \in \mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \rho_i (f(z_i') - f(z_i))\right]$$
$$\leq 2R_n(\mathcal{F})$$

where the right-hand expectation is taken over two independent samples $\mathbf{z}, \mathbf{z}' \sim P^n$ and the *Rademacher variables* $\rho_1, \ldots, \rho_n$. These are independent random variables, each with equal probability of taking the values -1 or 1, that give their name to the *Rademacher average*

$$R_n(\mathcal{F}) = \mathrm{E}\left[\sup_{f \in \mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \rho_l f(z_i)\right].$$

Intuitively, this quantity measures how well the functions in $\mathcal{F}$ can be chosen to align with randomly chosen labels ρ_i. The Rademacher averages for the loss class $\mathbb{L}$ and the hypothesis class $\mathcal{H}$ are closely related. For 0–1 loss, it can be shown they satisfy $R_n(\mathbb{L}) = \frac{1}{2}R_n(\mathcal{H})$.

Putting all the above steps together gives the following bounds.

Rademacher bound: Suppose $\mathcal{A}$ has hypothesis class $\mathcal{H}$. Then with probability at least $1 - \delta$ over draws of $\mathbf{z}$ from P^n the hypothesis $h = \mathcal{A}(\mathbf{Z})$ satisfies

$$L_P(h) \leq \hat{L}_z(h) + R_n(\mathcal{H}) + \sqrt{\frac{\ln\frac{1}{\delta}}{2n}}. \tag{8}$$

This bound is qualitatively different to the Growth Function and VC bounds above as the Rademacher average term is *distribution-dependent* whereas the other complexity terms are purely a function of the hypothesis space. Indeed, it is possible to bound the Rademacher average in terms of the VC dimension and obtain the VC bound (7) from (8). Furthermore, the Rademacher average is closely related to the minimum empirical risk via $R_n(\mathcal{H}) = 1 - 2\mathbb{E}[\inf_{h \in \mathcal{H}} \hat{L}_{\mathbf{x},\rho}(h)]$ where $\hat{L}_{\mathbf{x},\rho}(h)$ is the empirical risk of h for the randomly labeled sample $\mathbf{z} = ((x_1, \rho_1), \ldots, (x_n, \rho_n))$. Thus, in principle, $R_n(\mathcal{H})$ could be estimated for a given learning problem using standard ERM methods.

The Rademacher bound can be further refined so that the complexity term is *data-dependent* rather than distribution-dependent. This is done by noting that the Rademacher average $R_n\mathcal{F} = \mathbb{E}[\hat{R}_{\mathbf{z}}(\mathcal{F})]$ where $\hat{R}_{\mathbf{z}}(\mathcal{F})$ is the *empirical Rademacher average* for $\mathcal{F}$ conditioned on the sample $\mathbf{z}$. Applying McDiarmid's inequality to the difference between $\hat{R}_{\mathbf{z}}(\mathcal{F})$ and its mean gives a sample-dependent bound:

> *Empirical Rademacher bound*: Under the same conditions as the Rademacher bound, the following holds with probability $1-\delta$:
>
> $$L_P(h) \leq \hat{L}_z(h) + \hat{R}_z(\mathcal{H}) + 3\sqrt{\frac{\ln\frac{2}{\delta}}{2n}}. \qquad (9)$$

PAC-Bayesian bounds: All the bounds in the previous section provide bounds on deterministic hypotheses, which include complexity terms that are functions of the entire hypothesis space. PAC-Bayesian bounds differ from these in two ways: they provide bounds on nondeterministic hypotheses – labels may be predicted for instances stochastically; and their complexity terms are *hypothesis-dependent*. The term "Bayesian" given to these bounds refers to the use of a distribution over hypotheses that is used to define the complexity term. This distribution can be interpreted as a prior belief over the efficacy of each hypothesis before any observations are made.

Nondeterministic hypotheses are modeled by assuming that a distribution μ over $\mathcal{H}$ is used to randomly draw a deterministic hypothesis $h \in \mathcal{H}$ to predict $h(x)$ each time a new instance x is seen. Such a strategy is called a *Gibbs hypothesis* for μ. Since its behavior is defined by the distribution μ, we will abuse our notation slightly and define its loss on the example z to be $\ell_\mu(z) := \mathbb{E}_{h\sim\mu}[\ell_h(z)]$. Similarly, the true risk and empirical risk for a Gibbs hypothesis are, respectively, defined to be $L_P(\mu) := \mathbb{E}_{h\sim\mu}[L_P(h)]$ and $\hat{L}_{\mathbf{z}}(\mu) := \mathbb{E}_{h\sim\mu}[\hat{L}_{\mathbf{z}}(h)]$. As with the earlier generalization bounds, the aim is to provide guarantees about the difference between $L_P(\mu)$ and $\hat{L}_{\mathbf{z}}(\mu)$. In the case of 0–1 loss, $p := L_P(\mu) \in [0,1]$ is just the probability of the Gibbs hypothesis for μ misclassifying an example and $q := \hat{L}_{\mathbf{z}}(\mu) \in [0,1]$ can be thought of as an estimate of p. However, unlike the earlier bounds on the difference between the true and estimated risk, PAC-Bayesian bounds are expressed in terms the *Kullback–Leibler (KL) divergence*. For the values $p, q \in [0, 1]$ this is defined as $kl(q\|p) := q\ln\frac{q}{p} + (1-q)\ln\frac{1-q}{1-p}$ and for distributions μ and π over the hypothesis space $\mathcal{H}$ we write $KL(\mu \parallel \pi) := \int_{\mathcal{H}} \ln\frac{d\mu}{d\pi}d\mu$. Using these definitions, the most common PAC-Bayesian bound states the following.

> *Theorem (PAC-Bayesian bound)*: For all choices of the distribution π over $\mathcal{H}$ made prior to seeing any examples, the Gibbs hypothesis defined by μ satisfies
>
> $$kl(L_P(\mu), \hat{L}_{\mathbf{z}}(\mu)) \leq \frac{KL(\mu \parallel \pi) + \ln\frac{n+1}{\delta}}{n} \qquad (10)$$
>
> with probability at least 1 – δ over draws of $\mathbf{z}$ from P^n.

This says that the difference (as measured by *kl*) between the true and empirical risk for the Gibbs hypothesis based on μ is controlled by two terms: a *complexity* term $\frac{KL(\mu\|\pi)}{n}$ and a *sampling* term $\frac{\ln\frac{n+1}{\delta}}{n}$, both of which converge to zero as n increases. To make connections with the previous bounds more apparent, we can weaken (10) using the inequality $kl(q \parallel p) \geq 2(p-q)^2$ to get the following bound that holds under the same assumptions:

$$L_P(\mu) \leq \hat{L}_{\mathbf{z}}(\mu) + \sqrt{\frac{KL(\mu \parallel \pi) + \ln\frac{n+1}{\delta}}{2n}}.$$

The sampling term is similar to the ubiquitous estimation penalty in the earlier bounds but with an additional $\ln(n+1)/n$. The complexity term is a measure of the complexity of the Gibbs hypothesis for μ *relative* to the distribution π. Intuitively, $KL(\cdot \| \pi)$ can be thought of as a parametrized family of complexity measures where hypotheses from a region where π is large are "cheap" and those where π is small are "expensive". Information theoretically, it is the expected number of extra bits required to code hypotheses drawn from μ using a code based on π instead of a code based on μ. It is for these reasons the PAC-Bayes bound is said to demonstrate the importance of choosing a good prior. If the Gibbs hypothesis μ, which minimizes $\hat{L}_z(\mu)$ is also "close" to π then the bound will be tight.

Unlike the other bounds discussed above, PAC-Bayesian bounds are in terms of the complexity of single meta-classifiers rather than the complexity of classes. Furthermore, for specific base hypothesis classes such as margin classifiers used by SVMs it is possible to get hypothesis-specific bounds via the PAC-Bayesian bounds. These are typically much tighter than the VC or Rademacher bounds.

Other bounds: While the above bounds are landmarks in statistical learning theory there is obviously much more territory that has not been covered here. For starters, the VC bounds for classification can be refined by using more sophisticated results from empirical process theory such as the Bernstein and Variance-based bounds. These are discussed in Sect. 5 of Boucheron et al. (2005). There are also other distribution- and sample-dependent complexity measures that are motivated differently to Rademacher averages. For example, the *VC entropy* (see Sect. 4.5 of Bousquet et al. 2004) is a distribution-dependent measure obtained by averaging $|\mathcal{F}_{\mathbf{z}}|$ with respect to the sample distribution rather than taking supremum in the definition of the shattering coefficient.

Moving beyond classification, bounds for regression problems have been studied in depth and have similar properties to those for classification. These bounds are obtained by essentially discretizing the function spaces. The growth function is replaced by what is known as a *covering number* but the essence of the bounds remain the same. The reader is referred to Herbrich and Williamson (2002) for a brief discussion and Anthony and Bartlett (1999) for more detail.

There are a variety of bounds that, unlike those above, are algorithm-specific. For example, the regularized empirical risk minimization performed by SVMs has been analyzed within an *algorithmic stability* framework. As discussed in Boucheron et al. (2005) and Herbrich and Williamson (2002), hypotheses are considered stable if their predictions are not varied too much when a single training example is perturbed. Two other algorithm-dependent frameworks include the *luckiness* and *compression* frameworks, both summarized in Herbrich and Williamson (2002). The former gives bounds in terms of an a priori measure of luckiness – how well a training sample aligns with biases encoded in an algorithm – while the latter considers algorithms, like SVMs, which base hypotheses on key examples within a training sample.

Recently, there has been work on a type of algorithm-dependent, relative bound called *reductions* (see Beygelzimer et al. 2008 for an overview). By transforming inputs and outputs for one type of problem (e.g., probability estimation) into a different type of problem (e.g., classification), bounds for the former can be given in terms of bounds for the latter while making very few assumptions. This opens up a variety of avenues for applying existing results to new learning tasks.

Cross-References

- ▶ Classification
- ▶ Empirical Risk Minimization
- ▶ Hypothesis Space
- ▶ Loss
- ▶ PAC Learning
- ▶ Regression
- ▶ Regularization
- ▶ Structural Risk Minimization
- ▶ VC Dimension

Recommended Reading

As mentioned above, the uniform convergence bounds by Vapnik and Chervonenkis (1971) and the PAC framework of Valiant (1984) were the first generalization bounds for statistical learning. Ideas from both were synthesized and extended by Blumer et al. (1989). The book by Kearns and Vazirani (1994) provides a good overview of the early PAC-style bounds while Vapnik's comprehensive book (Vapnik 1998), and Anthony and Bartlett's book (1999) cover classification and regression bounds involving the VC dimension. Rademacher averages were first considered as an alternative to VC dimension in the context of learning theory by Koltchinskii and

Panchenko (2001) and were refined and extended by Bartlett and Mendelson (2003) who provide a readable overview. Early PAC-Bayesian bounds were established by McAllester (1999) based on an earlier PAC analysis of Bayesian estimators by Shawe-Taylor and Williamson (1997). Applications of the PAC-Bayesian bound to SVMs are discussed in Langford's tutorial on prediction theory (Langford 2005) and recent paper by Banerjee (2006) provides an information theoretic motivation, a simple proof of the bound in (11), as well as connections with similar bounds in online learning.

There are several well-written surveys of generalization bounds and learning theory in general. Herbrich and Williamson (2002) present a unified view of VC, compression, luckiness, PAC-Bayesian, and stability bounds. In a very readable introduction to statistical learning theory, Bousquet et al. (2004) provide good intuition and concise proofs for all but the PAC-Bayesian bounds presented above. That introduction is a good companion for the excellent but more technical survey by Boucheron et al. (2005) based on tools from the theory of empirical processes. The latter paper also provides a wealth of further references and a concise history of the development of main techniques in statistical learning theory.

Anthony M, Bartlett PL (1999) Neural network learning: theoretical foundations. Cambridge University Press, Cambridge

Banerjee A (2006) On Bayesian bounds. In: ICML'06: proceedings of the 23rd international conference on machine learning, Pittsburgh, pp 81–88

Bartlett PL, Mendelson S (2003) Rademacher and Gaussian complexities: risk bounds and structural results. J Mach Learn Res 3:463–482

Beygelzimer A, Langford J, Zadrozny B (2008) Machine learning techniques – reductions between prediction quality metrics. In: Zhen L, Cathy HX (eds) Performance modeling and engineering. Springer, New York, pp 3–28

Blumer A, Ehrenfeucht A, Haussler D, Warmuth MK (1989) Learnability and the Vapnik-Chervonenkis dimension. J ACM (JACM) 36(4):929–965

Boucheron S, Bousquet O, Lugosi G (2005) Theory of classification: a survey of some recent advances. ESAIM Probab Stat 9:323–375

Bousquet O, Boucheron S, Lugosi G (2004) Introduction to statistical learning theory. Volume 3176 of lecture notes in artificial intelligence. Springer, Berlin, pp 169–207

Herbrich R, Williamson RC (2002) Learning and generalization: theory and bounds. In: Arbib M (ed) Handbook of brain theory and neural networks, 2nd ed. MIT Press, Cambridge

Kearns MJ, Vazirani UV (1994) An introduction to computational learning theory. MIT Press, Cambridge

Koltchinskii V (2001) Rademacher penalties and structural risk minimization. IEEE Trans Inf Theory 47(5):1902–1914

Langford J (2005) Tutorial on practical prediction theory for classification. J Mach Learn Res 6(1):273–306

McAllester DA (1999) Some PAC-Bayesian theorems. Mach Learn 37(3):355–363

Shawe-Taylor J, Williamson RC (1997) A PAC analysis of a Bayesian estimator. In: Proceedings of the tenth annual conference on computational learning theory. ACM, New York, p 7

Valiant LG (1984) A theory of the learnable. Commun ACM 27(11):1142

Vapnik VN (1998) Statistical learning theory. Wiley, New York

Vapnik VN, Chervonenkis AY (1971) On the uniform convergence of relative frequencies of events to their probabilities. Theory Probab Appl 16(2):264–280

Generalization Performance

The *generalization performance* of a learning algorithm refers to the performance on ▸ out-of-sample data of the models learned by the algorithm.

Cross-References

▸ Algorithm Evaluation

Generalized Delta Rule

▸ Backpropagation

General-to-Specific Search

When searching a hypothesis space, a general-to-specific search starts from the most general hypothesis and expands the search by specialization. See ▸ Learning as Search.

Generative and Discriminative Learning

Bin Liu[1] and Geoffrey I. Webb[2]
[1]Monash University, Clayton, VIC, Australia
[2]Faculty of Information Technology, Monash University, Victoria, Australia

Definition

Generative learning refers alternatively to any classification learning process that classifies by using an estimate of the joint probability P(y,$\mathbf{x}$) or to any classification learning process that classifies by using estimates of the ▸ prior probability P(y) and the conditional probability P($\mathbf{x}|y$) (Jaakkola and Haussler 1999; Jaakkola et al. 1999; Ng and Jordan 2002; Lasserre et al. 2006; Bishop 2007), where y is a class and $\mathbf{x}$ is a description of an object to be classified. Given such models or estimates, it is possible to generate synthetic objects from the joint distribution. Generative learning contrasts to *discriminative learning* in which a model or estimate of P($y|\mathbf{x}$) is formed without reference to an explicit estimate of any of P($y,\mathbf{x}$), P($\mathbf{x}$), or P($\mathbf{x}|y$).

It is also common to categorize as discriminative approaches based on a decision function that directly map from input $\mathbf{x}$ onto the output y (such as ▸ support vector machines, ▸ neural networks, and ▸ decision trees), where the decision risk is minimized without estimation of P($y,\mathbf{x}$), P($\mathbf{x}|y$), or P($y|\mathbf{x}$) (Jaakkola and Haussler 1999).

The standard exemplar of generative learning is ▸ naïve Bayes and that of discriminative learning is ▸ logistic regression Another important contrasting pair is the generative ▸ hidden Markov model and discriminative ▸ conditional random field.

It is widely accepted that generative learning works well when samples are rare, while discriminative learning has better asymptotic error performance (Ng and Jordan 2002).

Motivation and Background

Efron (1975) provides an early examination of the generative/discriminative distinction. Efron performs an empirical comparison of the efficiency of the generative ▸ linear discriminant analysis (LDA) and discriminative ▸ logistic regression. His results show that logistic regression has 30 % less efficiency than LDA, which means the discriminative approach is 30 % slower to reach its asymptotic error than the generative approach.

Ng and Jordan (2002) give a theoretical discussion of the efficiency of generative ▸ naïve Bayes and discriminative ▸ logistic regression. This is an interesting pair because they both form linear models of forms that are directly equivalent to one another, the only substantive difference being the manner in which they parameterize those models. Their result shows that logistic regression converges toward its asymptotic error in order n samples, while naïve Bayes converges in order log$n$ samples. While logistic regression converges much slower than naïve Bayes, it has lower asymptotic error than naïve Bayes. These results suggest that it is desirable to use a generative approach when training data is scarce and to use a discriminative approach when there is enough training data. However, it is worth noting that the generative/discriminative distinction is not the only difference in how these two algorithms parameterize their models. Whereas logistic regression seeks to directly fit its model to the discriminative objective, P(y|$\mathbf{x}$), naïve Bayes does not directly fit P($y,\mathbf{x}$). Instead it fits its model to P(y) and each P(x$_i$|y) (where x$_i$ is an individual attribute), making the simplifying attribute independence assumption.

Recent research into the generative/discriminative learning distinction has concentrated on the area of hybrids of generative and discriminative learning as well as generative learning and discriminative learning in structured data learning or semi-supervised learning context.

In hybrid approaches, researchers seek to obtain the merits of both generative learning and discriminative learning. Some examples include the Fisher kernel for discriminative learning (Jaakkola and Haussler 1999), maxent discriminative learning (Jaakkola et al. 1999), and principled hybrids of generative and discriminative models (Lasserre et al. 2006; Zaidi et al. 2014).

In structured data learning, the output data have dependent relationships As an example of generative learning, hidden Markov models are used in structured data problems which need sequential decisions. The discriminative analogue is conditional random field models. Another example of discriminatively structured learning is max-margin Markov networks (Taskar et al. 2004).

In ▸ semi-supervised learning, co-training and multiview learning are usually applied to generative learning (Blum and Mitchell 1998). It is less straightforward to apply semi-supervised learning in traditional discriminative learning, since $P(y|\mathbf{x})$ is estimated by ignoring $P(\mathbf{x})$. Examples of semi-supervised learning methods in discriminative learning include transductive SVM, Gaussian processes, information regularization, and graph-based methods (Chapelle et al. 2006).

Recommended Reading

Bishop CM (2007) Pattern recognition and machine learning. Springer, New York

Blum A, Mitchell T (1998) Combining labeled and unlabeled data with co-training. In: Proceedings of the eleventh annual conference on computational learning theory

Chapelle O, Schölkopf B, Zien A (2006) Semi-supervised learning. The MIT Press, Cambridge

Efron B (1975) The efficiency of logistic regression compared to normal discriminant analysis. J Am Stat Assoc 70(352):892–898

Jaakkola TS, Haussler D (1999) Exploiting generative models in discriminative classifiers. Adv Neural Inf Process Syst 11:487–493

Jaakkola T, Meila M, Jebara T (1999) Maximum entropy discrimination. Adv Neural Inf Process Syst 12

Lasserre JA, Bishop CM, Minka TP (2006) Principled hybrids of generative and discriminative models. In: IEEE conference on computer vision and pattern recognition

Ng AY, Jordan MI (2002) On discriminative vs. generative classifiers: a comparison of logistic regression and naive Bayes. Adv Neural Inf Process Syst 2(14):841–848

Taskar B, Guestrin C, Koller D (2004) Max-margin Markov networks. Adv Neural Inf Process Syst 16

Zaidi N, Carman M, Webb GI (2014) Naive-Bayes inspired effective pre-conditioner for speeding-up logistic regression. In: Proceedings of the 14th IEEE international conference on data mining, ICDM-14, pp 1097–1102

Generative Learning

Definition

Generative learning refers alternatively to any classification learning process that classifies by using an estimate of the joint probability $P(y, \mathbf{x})$ or to any classification learning process that classifies by using estimates of the prior probability P(y) and the conditional probability $P(\mathbf{x}|y)$, where y is a class and $\mathbf{x}$ is a description of an object to be classified. Given such models or estimates it is possible to generate synthetic objects from the joint distribution. Generative learning contrasts to discriminative learning in which a model or estimate of $P(y|\mathbf{x})$ is formed without reference to an explicit estimate of any of $P(\mathbf{x}), P(y, \mathbf{x})$, or $P(\mathbf{x}|y)$.

Cross-References

▸ Generative and Discriminative Learning

Genetic and Evolutionary Algorithms

Claude Sammut
The University of New South Wales, Sydney,
NSW, Australia

Definitions

There are many variations of genetic algorithms (GA). Here, wedescribe a simple scheme to introduce some of the key terms in genetic and evolutionary algorithms. See the main entry on ▸ Evolutionary Algorithms for references to specific methods.

In genetic learning, we assume that there is a population of individuals, each of which represents a candidate problem solver for a given task. GAs can be thought of as a family of general purpose search methods that are capable of solving a broad range of problems from optimization and scheduling to robot control. Like evolution, genetic algorithms test each individual from the population and only the fittest survive to reproduce for the next generation. The algorithm creates new generations until at least one individual is found that can solve the problem adequately.

Each problem solver is a *chromosome*. A position, or set of positions in a chromosome is called a *gene*. The possible values (from a fixed set of symbols) of a gene are known as *alleles*. For example, a simple genetic algorithm may define the set of symbols to be {0, 1}, and chromosome lengths are fixed. The most critical problem in applying a genetic algorithm is in finding a suitable encoding of the examples in the problem domain to a chromosome. A good choice of representation will make the search easier by limiting the size of the search space. A poor choice will result in a large search space. Choosing the size of the population can be problematic since a small population size provides an insufficient sample over the space of solutions for a problem and large population requires extensive evaluation and will be slow.

Each iteration in a genetic algorithm is called a *generation*. Each chromosome in a population is used to solve a problem. Its performance is evaluated and the chromosome is given a rating of fitness. The population is also given an overall fitness rating based on the performance of its members. The fitness value indicates how close a chromosome or population is to the required solution.

New sets of chromosomes are produced from one generation to the next. Reproduction takes place when selected chromosomes from one generation are recombined with others to form chromosomes for the next generation. The new ones are called *offspring*. Selection of chromosomes for reproduction is based on their fitness values. The average fitness of the population may also be calculated at the end of each generation. The strategy must be modified if too few or too many chromosomes survive. For example, at least 10 % and at most 60 % must survive.

Genetic Operators

Operators that recombine the selected chromosomes are called *genetic operators*. Two common operators are *crossover* and *mutation*. Crossover exchanges portions of a pair of chromosomes at a randomly chosen point called the crossover point. Some Implementations have more than one crossover point. For example, if there are two chromosomes, X and Y:

$$X = 1001\,01011, Y = 1110\,10010$$

and the crossover point is after position 4, the resulting offspring are:

$$O1 = 100110010, O2 = 1110\,01011$$

Offspring produced by crossover cannot contain information that is not already in the population, so an additional operator, *mutation*, is required. Mutation generates an offspring by randomly changing the values of genes at one or more gene positions of a selected chromosome. For example, if the following chromosome,

$$Z = 100101011$$

is mutated at positions 2, 4, and 9, then the resulting offspring is:

$$O = 110001010$$

The number of offspring produced for each new generation depends on how members are introduced so as to maintain a fixed population size. In a *pure* replacement strategy, the whole population is replaced by a new one. In an *elitist* strategy, a proportion of the population survives to the next generation.

Cross-References

▶ Evolutionary Algorithms

Genetic Attribute Construction

▸ Evolutionary Feature Selection and Construction

Genetic Clustering

▸ Evolutionary Clustering

Genetic Feature Selection

▸ Evolutionary Feature Selection and Construction

Genetic Grouping

▸ Evolutionary Clustering

Genetic Neural Networks

▸ Neuroevolution

Genetic Programming

Moshe Sipper
Ben-Gurion University, Beer-Sheva, Israel

Abstract

Genetic programming (GP) is an evolutionary algorithm-based methodology inspired by biological evolution, used to solve complex problems.

Genetic programming is a subclass of ▸ evolutionary algorithms, wherein a population of individual programs is evolved. The main mechanism behind genetic programming is that of a ▸ genetic algorithm, namely, the repeated cycling through four operations applied to the entire population: evaluate–select–crossover–mutate. Starting with an initial population of randomly generated programs, each individual is evaluated in the domain environment and assigned a fitness value representing how well the individual solves the problem at hand. Being randomly generated, the first-generation individuals usually exhibit poor performance. However, some individuals are better than others, that is, as in nature, variability exists, and through the mechanism of selection, these have a higher probability of being selected to parent the next generation. The size of the population is finite and usually constant.

See ▸ Evolutionary Games for a more detailed explanation of genetic programming.

Genetics-Based Machine Learning

▸ Classifier Systems

Gibbs Sampling

Gibbs Sampling is a heuristic inference algorithm for ▸ Bayesian networks. See ▸ Graphical Models for details.

Gini Coefficient

The Gini coefficient is an empirical measure of classification performance based on the area under an ROC curve (AUC). Attributed to the Italian statistician Corrado Gini (1884–1965), it can be calculated as $2 \cdot \{\text{AUC}\} - 1$ and thus takes values in the interval $[-1, 1]$, where 1 indicates perfect ranking performance and -1 indicates that all negatives are ranked before all positives. See ▸ ROC Analysis.

Gram Matrix

▶ Kernel Matrix

Grammar Learning

▶ Grammatical Inference

Grammatical Inference

Lorenza Saitta[1] and Michele Sebag[2]
[1]Università del Piemonte Orientale, Alessandria, Italy
[2]CNRS – INRIA – Université Paris-Sud, Orsay, France

Synonyms

Grammar learning

Definition

Grammatical inference is concerned with inferring grammars from positive (and possibly negative) examples (Angluin 1978; Korfiatis and Paliouras 2008; Sakakibara 2005). A context-free grammar (CFG) $\mathcal{G}$ (equivalent to a push-down finite-state automaton) is described by a four tuple $(\mathcal{Q}, \mathcal{E}, \delta, \Sigma)$:

- Σ is the alphabet of *terminal* symbols, upon which the grammar is defined.
- The pair $(\mathcal{Q}, \mathcal{E})$ defines a graph, where $\mathcal{Q}$ is the set of nodes (states), and $\mathcal{E}$ is the set of edges (production rules). $\mathcal{Q}$ includes one *starting* node q_0 and a set $\mathcal{Q}_f$ ($\mathcal{Q}_f \subset \mathcal{Q}$) of final or *accepting* nodes.
- Every edge in $\mathcal{E}$ is labeled by one or several letters in Σ, expressed through mapping $\delta : \mathcal{E} \mapsto 2^{\Sigma}$.
- Let $\mathcal{L}(\mathcal{G})$ denote the language associated to the grammar. Each string s in $\mathcal{L}(\mathcal{G})$ is generated along a random walk in the graph, starting in q_0 with an initially empty s. Upon traversing edge e, one symbol from $\delta(e)$ is concatenated to s. The walk ends upon reaching a final node ($e \in \mathcal{Q}_f$).

A CFG is determinist if all pairs of edges (q, q') and (q, q'') $(q' \neq q'')$ bear different labels $(\delta(q, q') \bigcap \delta(q, q'') = \emptyset)$.

One generalizes a given CFG by applying one or several operators, among the following: (1) introducing additional nodes and edges, (2) turning a node into an accepting one, and (3) *merging* two nodes q and q'. In the latter case, some non-determinism can be introduced (if some edges (q, r) and (q', r') have label(s) in common); enforcing a deterministic generalization is done using the *recursive determinization operator* (e.g., merging nodes r and r').

In general, grammatical inference proceeds as follows (Lang et al. 1998; Oncina and Garcia 1992). Let S be the set of positive examples, strings on alphabet Σ. The *prefix tree acceptor* (PTA), a most specific generalization of S, is constructed by associating to each character of every string a distinct node and applying the determinization operator. This PTA is thereafter iteratively generalized by merging a pair of nodes. Well-known grammar learners are RPNI (Oncina and Garcia 1992) and BLUE-FRINGE (Lang et al. 1998). RPNI uses a depth first search strategy and merges the pair of nodes which are closest to the start node, such that their deterministic generalization does not cover any negative example. BLUE-FRINGE uses a beam search from a candidate list, selecting the pair of nodes to be merged after the *evidence-driven* state merging (EDSM) criterion, i.e., such that their generalization involves a minimal number of final states.

Recommended Reading

Angluin D (1978) On the complexity of minimum inference of regular sets. Inf Control 39:337–350

Korfiatis G, Paliouras G (2008) Modeling web navigation using grammatical inference. Appl Artif Intell 22(1–2):116–138

Lang KJ, Pearlmutter BA, Price RA (1998) Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In: ICGI'98: proceedings of the 4th international colloquium on grammatical inference. Springer, Berlin, pp 1–12
Oncina J, Garcia P (1992) Inferring regular languages in polynomial update time. In: Pattern recognition and image analysis, vol 1. World Scientific, Singapore/New Jersey, pp 49–61
Sakakibara Y (2005) Grammatical inference in bioinformatics. IEEE Trans Pattern Anal Mach Intell 27(7):1051–1062

Grammatical Tagging

▸ POS Tagging

Graph Clustering

Charu C. Aggarwal
IBM T. J. Watson Research Center, Hawthorne, NY, USA

Synonyms

Minimum cuts; Network clustering; Spectral clustering; Structured data clustering

Definition

Graph clustering refers to ▸ clustering of data in the form of graphs. Two distinct forms of clustering can be performed on graph data. Vertex clustering seeks to cluster the nodes of the graph into groups of densely connected regions based on either edge weights or edge distances. The second form of graph clustering treats the graphs as the objects to be clustered and clusters these objects on the basis of similarity. The second approach is often encountered in the context of structured or XML data.

Motivation and Background

Graph clustering is a form of ▸ graph mining that is useful in a number of practical applications including marketing, customer segmentation, congestion detection, facility location, and XML data integration (Lee et al. 2002). The graph clustering problems are typically defined into two categories:

- *Node clustering algorithms*: Node clustering algorithms are generalizations of multidimensional clustering algorithms in which we use functions of the multidimensional data points in order to define the distances. In the case of graph clustering algorithms, we associate numerical values with the edges. These numerical values need not satisfy traditional properties of distance functions such as the triangle inequality. We use these distance values in order to create clusters of nodes. We note that the numerical value associated with a given node may either be a distance value or a similarity value. Correspondingly, the objective function associated with the partitioning may either be minimized or maximized. We note that the problem of minimizing the intercluster similarity for a fixed number of clusters essentially reduces to the problem of *graph partitioning* or the *minimum multiway cut problem*. This is also referred to the problem of mining dense graphs and pseudocliques. Recently, the problem has also been studied in the database literature as that of *quasi-clique determination*. In this problem, we determine groups of nodes which are "almost cliques." In other words, an edge exists between any pair of nodes in the set with a high probability. A closely related problem is that of determining *shingles* (Gibson et al. 2005). Shingles are defined as those subgraphs which have a large number of common links. This is particularly useful for massive graphs which contain a large number of nodes. In such cases, a min-hash approach (Gibson et al. 2005) can be used in order to summarize the structural behavior of the underlying graph.

- *Graph clustering algorithms*: In this case, we have a (possibly large) number of graphs which need to be clustered based on their underlying structural behavior. This problem is challenging because of the need to match the structures of the underlying graphs and use these structures for clustering purposes. Such algorithms are discussed both in the context of classical graph data sets as well as semistructured data. In the case of semistructured data, the problem arises in the context of a large number of documents which need to be clustered on the basis of the underlying structure and attributes. It has been shown by Aggarwal et al. (2007) that the use of the underlying document structure leads to significantly more effective algorithms.

This chapter will discuss the different kinds of clustering algorithms and their applications. Each section will discuss a particular class of clustering algorithms and the different approaches which are commonly used for this class.

Graph Clustering as Minimum Cut

The graph clustering problem can be related to the minimum-cut and graph partitioning problems. In this case, it is assumed that the underlying graphs have weights on the edges. It is desired to partition the graphs in such a way so as to minimize the weights of the edges across the partitions. In general, we would like to partition the graphs into k groups of nodes. However, since the special case $k = 2$ is efficiently solvable, we would like to first provide a special discussion for this case. This version is polynomially solvable, since it is the mathematical dual of the maximum-flow problem. This problem is also referred to as the *minimum-cut problem*.

The minimum-cut problem is defined as follows. Consider a graph $G = (N, A)$ with node set N and edge set A. The node set N contains the source s and sink t. Each edge $(i, j) \in A$ has a weight associated with it which is denoted by u_{ij}. We note that the edges may be either undirected or directed, though the undirected case is often much more relevant for connectivity applications. We would like to partition the node set N into two groups S and $N - S$. The set of edges such that one end lies in S and the other lies in $N - S$ is denoted by $C(S, N - S)$. We would like to partition the node set N into two sets S and $N - S$, such that the sum of the weights in $C(S, N - S)$ is minimized. In other words, we would like to minimize $\sum_{(i,j)} \in C_{(S,N-S)} u_{ij}$. This is the unrestricted version of the minimum-cut problem. We will examine two variations of the minimum-cut problem:

- We wish to determine the global minimum $s - t$ cut with no restrictions on the membership of nodes to different partitions.
- We wish to determine the minimum $s - t$ cut, in which one partition contains the source node s and the other partition contains the sink node t.

It is easy to see that the former problem can be solved by using repeated applications of the latter algorithm. By fixing s and choosing different values of the sink t, it can be shown that the global minimum cut may be effectively determined.

It turns out that the maximum-flow problem is the mathematical dual of the minimum-cut problem. In the maximum-flow problem, we assume that the weight u_{ij} is a capacity of the edge (i, j). Each edge is allowed to have a *flow*x_{ij} which is at most equal to the capacity u_{ij}. Each node other than the source s and sink t is assumed to satisfy the *flow conservation property*. In other words, for each node $i \in N$ we have

$$\sum_{j:(i,j)\in A} x_{ij} = \sum_{j:(j,i)\in A} x_{ji}.$$

We would like to maximize the total flow originating from the source and reaching the sink t, subject to the above constraints. The maximum-flow problem is solved with the use of a variety of *augmenting path* and *preflow push algorithms*. Details of different kinds of algorithms may be found in the work by Ahuja et al. (1992).

A closely related problem to the minimum $s - t$ cut problem is that of determining a *global minimum cut* in an undirected graph. This particular case is more efficient than that of finding the $s - t$ minimum cut. One way of determining a minimum cut is by using a contraction-based edge-sampling approach. While the previous technique is applicable to both the directed and undirected versions of the problem, the contraction-based approach is applicable only to the undirected version of the problem. Furthermore, the contraction-based approach is applicable only for the case in which the weight of each edge is $u_{ij} = 1$. While the method can easily be extended to the weighted version by varying the edge-sampling probability, the polynomial running time bounds discussed by Tsay et al. (1999) do not apply to this case. The contraction approach is a probabilistic technique in which we successively sample the edges in order to collapse nodes into larger sets of nodes. By successively sampling different sequences of edges and picking the optimum value (Tsay et al. 1999), it is possible to determine a global minimum cut. The broad idea of the contraction-based approach is as follows. We pick an edge randomly in the graph and contract its two end points into a single node. We remove all the self-loops which are created as a result of the contraction. We may also create some parallel edges, which are allowed to remain, since they influence the sampling probability (Alternatively, we may replace parallel edges by a single edge of weight which is equal to the number of parallel edges. We use this weight in order to bias the sampling process.) of contractions. The process of contraction is repeated until we are left with two nodes. We note that each of this pair of "super-nodes" corresponds to a set of nodes in the original data. These two sets of nodes provide us with the final minimum cut. We note that the minimum cut will survive in this approach, if none of the edges in the minimum cut are sampled during the contraction. It has been shown by Tsay et al. that by using repeated contraction of the graph to a size of $\sqrt{n}$ nodes, it is possible to obtain a correct solution with high probability in $O(n^2)$ time.

Graph Clustering as Multiway Graph Partitioning

The *multiway graph partitioning problem* is significantly more difficult, and is NP-hard (Kernighan and Lin 1970). In this case, we wish to partition a graph into $k > 2$ components, so that the total weight of the edges whose ends lie in different partitions is minimized. A well-known technique for graph partitioning is the Kerninghan-Lin algorithm (Kernighan and Lin 1970). This classical algorithm is based on hill climbing (or more generally neighborhood-search technique) for determining the optimal graph partitioning. Initially, we start off with a random cut of the graph. In each iteration, we exchange a pair of vertices in two partitions to see if the overall cut value is reduced. In the event that the cut value is reduced, then the interchange is performed. Otherwise, we pick another pair of vertices in order to perform the interchange. This process is repeated until we converge to a optimal solution. We note that this optimum may not be a global optimum, but may only be a local optimum of the underlying data. The main variation in different versions of the Kerninghan-Lin algorithm is the policy which is used for performing the interchanges on the vertices. Some examples of strategies which may be used in order to perform the interchange are as follows:

- We randomly pick a pair of vertices and perform the interchange, if it improves the underlying solution quality.
- We test all possible vertex-pair interchanges (or a sample of possible interchanges), and pick the interchange which improves the solution by the greatest amount.
- A k-interchange is one in which a sequence of k interchanges are performed at one time. We can test any k-interchange and perform it, if it improves the underlying solution quality.
- We can pick the optimal k-interchange from a sample of possibilities.

We note that the use of more sophisticated strategies allows a better improvement in the objective function for each interchange, but also requires more time for each interchange. For example, the determination of an optimal k-interchange requires much more time than a straightforward interchange. This is a natural trade-off which may work out differently depending upon the nature of the application at hand. Furthermore, the choice of the policy also affects the likelihood of getting stuck at a local optimum. For example, the use of k-interchange techniques are far less likely to result in local optimum for larger values of k. In fact, by choosing the best interchange across all possible values of k it is possible to ensure that a global optimum is always reached. On the other hand, it is increasingly difficult to implement the algorithm efficiently with increasing value of k. This is because the time complexity of the interchange increases exponentially with the value of k.

Graph Clustering with k-Means

Two well-known (and related) techniques for clustering in the context of multidimensional data (Jain and Dubes 1998) are the k-medoid and k-means algorithms. In the k-medoid algorithm (for multidimensional data), we sample a small number of points from the original data as *seeds* and assign every other data point from the clusters to the closest of these seeds. The closeness may be defined based on a user-defined objective function. The objective function for the clustering is defined as the sum of the corresponding distances of data points to the corresponding seeds. In the next iteration, the algorithm interchanges one of the seeds for another randomly selected seed from the data, and checks if the quality of the objective function improves upon performing the interchange. If this is indeed the case, then the interchange is accepted. Otherwise, we do not accept the interchange and try another sample interchange. This process is repeated, until the objective function does not improve over a predefined number of interchanges. A closely related method is the k-means method. The main difference with the k-medoid method is that we do not use representative points from the original data after the first iteration of picking the original seeds. In subsequent iterations, we use the centroid of each cluster as the seed set for the next iteration. This process is repeated until the cluster membership stabilizes.

A method has been proposed by Rattigan et al. (2007), which uses the characteristics of both the k-means and k-medoids algorithms. As in the case of the conventional partitioning algorithms, it picks k graph nodes as seeds. The main differences from the conventional algorithms are in terms of computation of distances (for assignment purposes), and in determination of subsequent seeds. A natural distance function for graphs is the *geodesic distance*, or the smallest number of hops between a pair of nodes. In order to determine the seed set for the next iteration, we compute the *local closeness centrality* for each cluster, and use the corresponding node as the sample seed. Thus, while this algorithm continues to use seeds from the original data set (as in the k-medoids algorithm), it uses intuitive ideas from the k-means algorithms in order to determine the identity of these seeds.

Graph Clustering with the Spectral Method

Eigenvector techniques are often used in multidimensional data in order to determine the underlying correlation structure in the data. It is natural to question as to whether such techniques can also be used for the more general case of graph data. It turns out that this is indeed possible with the use of a method called *spectral clustering*.

In the spectral clustering method, we make use of the node-node adjacency matrix of the graph. For a graph containing n nodes, let us assume that we have an $n \times n$ adjacency matrix, in which the entry (i, j) correspond to the weight of the edge between the nodes i and j. This essentially corresponds to the similarity between nodes i and j. This entry is denoted by w_{ij}, and the corresponding matrix is denoted by W. This matrix is assumed to be symmetric, since we are working with undirected graphs. Therefore, we assume that $w_{ij} = w_{ji}$ for any pair (i, j). All diagonal entries of the matrix W are assumed to be 0. As discussed earlier, the aim of any node partitioning algorithm is to minimize (a function of) the weights across the partitions. The spectral clustering method constructs this minimization function in terms of the matrix structure of the adjacency matrix and another matrix which is referred to as the *degree matrix*.

The *degree matrix D* is simply a diagonal matrix in which all entries are zero except for the diagonal values. The diagonal entry d_{ii} is equal to the sum of the weights of the incident edges. In other words, the entry d_{ij} is defined as follows:

$$d_{ij} = \sum_{j=1}^{n} w_{ij}, \quad i = j,$$

$$0, \quad i \neq j.$$

We formally define the *Laplacian matrix* as follows: (*Laplacian matrix*): The Laplacian matrix L is defined by subtracting the weighted adjacency matrix from the degree matrix. In other words, we have

$$L = D - W.$$

This matrix encodes the structural behavior of the graph effectively and its eigenvector behavior can be used in order to determine the important clusters in the underlying graph structure. It can be shown that the Laplacian matrix L is positive semidefinite i.e., for any n-dimensional row vector $f = [f_1 \dots fn]$ we have $f \cdot L \cdot f^T \geq 0$. This can be easily shown by expressing L in terms of its constituent entries which are a function of the corresponding weights w_{ij}. Upon expansion, it can be shown that

$$f \cdot L \cdot f^{\mathrm{T}} = (1/2) \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \cdot (f_i - f_J)^2.$$

The Laplacian matrix L is positive semidefinite. Specifically, for any n-dimensional row vector $f = [f_1 \dots f_n]$, we have

$$f \cdot L \cdot f^T = (1/2) \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \cdot (f_i - f_J)^2.$$

At this point, let us examine some *interpretations* of the vector f in terms of the underlying graph partitioning. Let us consider the case in which each f_i is drawn from the set $\{0, 1\}$, and this determines a two-way partition by labeling each node either 0 or 1. The particular partition to which the node i belongs is defined by the corresponding label. Note that the expansion of the expression $f \cdot L \cdot f^T$ from the above relationship simply represents the sum of the weights of the edges across the partition defined by f. Thus, the determination of an appropriate value of f for which the function $f \cdot L \cdot f^T$ is minimized also provides us with a good node partitioning. Unfortunately, it is not easy to determine the *discrete values* of f which determine this optimum partitioning. Nevertheless, we will see later in this section that even when we restrict f to real values, this provides us with the intuition necessary to create an effective partitioning.

An immediate observation is that the indicator vector $f = [1 \dots 1]$ is an eigenvector with a corresponding eigenvalue of 0. We note that $f = [1 \dots 1]$ must be an eigenvector, since L is positive semidefinite and $f \cdot L \cdot f^T$ can be 0 only for eigenvectors with 0 eigenvalues. This observation can be generalized further in order to determine the number of connected components in the graph. We make the following observation.

The number of (*linearly independent*) *eigenvectors with zero eigenvalues for the Laplacian matrix L is equal to the number of connected components in the underlying graph.*

We observe that connected components are the most obvious examples of clusters in the graph. Therefore, the determination of eigenvectors corresponding to zero eigenvalues provides us the information about (relatively rudimentary set of) clusters. Broadly speaking, it may not be possible to glean such clean membership behavior from the other eigenvectors. One of the problems is that other than this particular rudimentary set of eigenvectors (which correspond to the connected components), the vector components of the other eigenvectors are drawn from the real domain rather than the discrete {0, 1} domain. Nevertheless, because of the nature of the natural interpretation of $f \cdot L \cdot f^T$ in terms of the weights of the edges across nodes with very differing values of f_i, it is natural to cluster together the nodes for which the values of f_i are as similar as possible across any particular eigenvector on an average. This provides us with the intuition necessary to define an effective spectral clustering algorithm, which partitions the data set into k clusters for any arbitrary value of k. The algorithm is as follows:

- Determine the k eigenvectors with the smallest eigenvalues. Note that each eigenvector has as many components as the number of nodes. Let the component of the jth eigenvector for the ith node be denoted by p_{ij}.
- Create a new data set with as many records as the number of nodes. The ith record in this data set corresponds to the ith node and has k components. The record for this node is simply the eigenvector components for that node, which are denoted by $pi_1 \ldots p_{ik}$.
- Since we would like to cluster nodes with similar eigenvector components, we use any conventional clustering algorithm (e.g., k-means) in order to create k clusters from this data set. Note that the main focus of the approach was to create a *transformation* of a structural clustering algorithm into a more conventional multidimensional clustering algorithm, which is easy to solve. The particular choice of the multidimensional clustering algorithm is orthogonal to the broad spectral approach.

The above algorithm provides a broad framework for the spectral clustering algorithm. The input parameter for the above algorithm is the number of clusters k. In practice, a number of variations are possible in order to tune the quality of the clusters which are found. More details on the different methods which can be used for effective spectral graph clustering may be found in Chung (1997).

Graph Clustering as Quasi-clique Detection

A different way of determining massive graphs in the underlying data is that of determining *quasi-cliques*. This technique is different from many other partitioning algorithms, in that it focuses on definitions which maximize the edge densities *within a partition*, rather than minimizing the edge densities across partitions. A clique is a graph in which every pair of nodes are connected by an edge. A quasi-clique is a relaxation on this concept, and is defined by imposing a lower bound on the degree of each vertex in the given set of nodes. Specifically, we define a γ-quasi-clique is as follows:

A k-graph ($k \geq 1$) G is a γ-quasi-clique if the degree of each node in the corresponding subgraph of vertices is at least $\gamma \cdot k$.

The value of γ always lies in the range (0, 1]. We note that by choosing $\gamma = 1$, this definition reverts to that of standard cliques. Choosing lower values of γ allows for the relaxations which are more true in the case of real applications. This is because we rarely encounter complete cliques in real applications, and at least some edges within a dense subgraph would always be missing. A vertex is said to be critical if its degree in the corresponding subgraph is the smallest integer which is at least equal to $\gamma \cdot k$.

The earliest piece of work on this problem is from Abello et al. (2002). The work of Abello et al. (2002) uses a greedy randomized adaptive search algorithm, GRASP, to find a quasi-clique with the maximum size. A closely related problem is that of finding *frequently occurring cliques* in *multiple data sets*. In other words, when multiple graphs are obtained from different data sets, some dense subgraphs occur frequently together in the different data sets. Such graphs help in determining *important dense patterns of behavior in different data sources*. Such techniques find applicability in mining important patterns in graphical representations of customers. The techniques are also helpful in mining cross-graph quasi-cliques in gene expression data. An efficient algorithm for determining cross graph quasi-cliques was proposed by Pei et al. (2005).

The main restriction of the work proposed by Pei et al. (2005) is that the support threshold for the algorithms is assumed to be 100 %. This restriction has been relaxed in subsequent work (Zeng et al. 2007). The work by Zeng et al. (2007) examines the problem of mining frequent, closed quasi-cliques from a graph database with arbitrary support thresholds.

Graph Clustering as Dense Subgraph Determination

A closely related problem is that of dense subgraph determination in massive graphs. This problem is frequently encountered in large graph data sets. For example, the problem of determining large subgraphs of web graphs was studied by Gibson et al. (2005). The broad idea in the min-hash approach is to represent the outlinks of a particular node as sets. Two nodes are considered similar if they share many outlinks. Thus, consider a node A with an outlink set S_A, and a node B with outlink set S_B. Then the similarity between the two nodes is defined by the *Jaccard coefficient*, which is defined as $\frac{S_A \cap S_B}{S_A \cup S_B}$. We note that explicit enumeration of all the edges in order to compute this can be computationally inefficient. Rather, a *min-hash approach* is used in order to perform the estimation. This *min-hash approach* is as follows. We sort the universe of nodes in a random order. For any set of nodes in random sorted order, we determine the first node $First(A)$ for which an outlink exists from A to $First(A)$. We also determine the first node $First(B)$ for which an outlink exists from B to $First(B)$. It can be shown that the Jaccard coefficient is an unbiased estimate of the probability that $First(A)$ and $First(B)$ are the same nodes. By repeating this process over different permutations over the universe of nodes, it is possible to accurately estimate the Jaccard coefficient. This is done by using a constant number of permutations c of the node order. The actual permutations are implemented by associated c different randomized hash values with each node. This creates c sets of hash values of size n. The sort-order for any particular set of hash-values defines the corresponding permutation order. For each such permutation, we store the minimum node index of the outlink set. Thus, for each node, there are c such minimum indices. This means that, for each node, a fingerprint of size c can be constructed. By comparing the fingerprints of two nodes, the Jaccard coefficient can be estimated. This approach can be further generalized with the use of every s element set contained entirely with S_A and S_B. Thus, the above description is the special case when s is set to 1. By using different values of s and c, it is possible to design an algorithm which distinguishes between two sets that are above or below a certain threshold of similarity.

The overall technique by Gibson et al. (2005) first generates a set of c shingles of size s for each node. The process of generating the c shingles is extremely straightforward. Each node is processed independently. We use the min-wise hash function approach in order to generate subsets of size s from the outlinks at each node. This results in c subsets for each node. Thus, for each node, we have a set of c shingles. Thus, if the graph contains a total of n nodes, the total size of this shingle fingerprint is $n \times c \times sp$, where sp is the space required for each shingle. Typically, sp will be $O(s)$, since each shingle contains s nodes. For each distinct shingle thus created, we can create a list of nodes which contain it. In general, we would like to determine groups of shingles which contain a large number of common nodes. In order to do so, the method by Gibson et al. performs a second-order shingling in which the meta-shingles are created from the shingles. Thus, this further compresses the graph in a data structure of size $c \times c$. This is essentially a constant-size data structure. We note that this group of meta-shingles have the property that they contain a large number of common nodes. The dense subgraphs can then be extracted from these meta-shingles. More details on this approach may be found in the work by Gibson et al.

Clustering Graphs as Objects

In this section, we will discuss the problem of clustering *entire graphs* in a *multigraph database*, rather than examining the node clustering problem within a single graph. Such situations are often encountered in the context of XML data, since each XML document can be regarded as a structural record, and it may be necessary to create clusters from a large number of such objects. We note that XML data is quite similar to graph data in terms of how the data is organized structurally. The attribute values can be treated as graph labels and the corresponding semistructural relationships as the edges. In has been shown by Aggarwal et al. (2007), Dalamagas et al. (2005), Lee et al. (2002), and Lian et al. (2004) that this structural behavior can be leveraged in order to create effective clusters.

Since we are examining entire graphs in this version of the clustering problem, the problem simply boils down to that of clustering arbitrary *objects*, where the objects in this case have structural characteristics. Many of the conventional algorithms discussed by Jain and Dubes (1998) (such as k-means type partitional algorithms and hierarchical algorithms) can be extended to the case of graph data. The main changes required in order to extend these algorithms are as follows:

- Most of the underlying classical algorithms typically use some form of distance function in order to measure similarity. Therefore, we need appropriate measures in order to define similarity (or distances) between structural objects.
- Many of the classical algorithms (such as k-means) use *representative objects* such as centroids in critical intermediate steps. While this is straightforward in the case of multidimensional objects, it is much more challenging in the case of graph objects. Therefore, appropriate methods need to be designed in order to create representative objects. Furthermore, in some cases it may be difficult to create representatives in terms of single objects. We will see that it is often more robust to use *representative summaries* of the underlying objects.

There are two main classes of conventional techniques, which have been extended to the case of structural objects. These techniques are as follows:

- *Structural distance-based approach*: This approach computes structural distances between documents and uses them in order to compute clusters of documents. One of the earliest work on clustering tree structured data is the *XClust algorithm* (Lee et al. 2002), which was designed to cluster XML schemas in order for efficient integration of large numbers of document type definitions (DTDs) of XML sources. It adopts the agglomerative hierarchical clustering method which starts with clusters of single DTDs and gradually merges the two most similar clusters into one larger cluster. The similarity between two DTDs is based on their element similarity, which can be computed according to the semantics, structure, and context information of the elements in the corresponding DTDs. One of the shortcomings of the XClust algorithm is that it does not make full use of the structure information of the DTDs, which is quite important in the context of clustering tree-like structures. The method by Chawathe (1999) computes similarity measures based on the structural edit-distance between documents. This edit-distance is used in order to compute the distances between clusters of documents.

  S-GRACE is hierarchical clustering algorithm (Lian et al. 2004). In the work by Lian et al., an XML document is converted to a structure graph (or s-graph), and the distance between two XML documents is defined according to the number of the common element-subelement relationships, which can capture better structural similarity relationships than the tree edit-distance in some cases (Lian et al.).

- *Structural summary-based approach*: In many cases, it is possible to create summaries from the underlying documents. These summaries are used for creating groups of documents which are similar to these summaries. The first summary-based approach for clustering XML documents was presented by Dalamagas et al. (2005). In the work by Dalamagas et al., the XML documents are modeled as rooted, ordered labeled trees. A framework for clustering XML documents by using structural summaries of trees is presented. The aim is to improve algorithmic efficiency without compromising cluster quality.

  A second approach for clustering XML documents is presented by Aggarwal et al. (2007). This technique is a partition-based algorithm. The primary idea in this approach is to use frequent-pattern mining algorithms in order to determine the summaries of frequent structures in the data. The technique uses a k-means type approach in which each cluster center comprises a set of frequent patterns which are local to the partition for that cluster. The frequent patterns are mined using the documents assigned to a cluster center in the last iteration. The documents are then further reassigned to a cluster center based on the average similarity between the document and the newly created cluster centers from the local frequent patterns. In each iteration the document assignment and the mined frequent patterns are iteratively reassigned until the cluster centers and document partitions converge to a final state. It has been shown by Aggarwal et al. that such a structural summary-based approach is significantly superior to a similarity function-based approach, as presented by Chawathe (1999). The method is also superior to the structural approach by Dalamagas et al. (2005) because of its use of more robust representations of the underlying structural summaries.

Conclusions and Future Research

In this chapter, we presented a review of the commonly known algorithms for clustering graph data. The problem of clustering graphs has been widely studied in the literature, because of its application to a variety of data mining and data management problems. Graph clustering algorithms are of two types:

- *Node clustering algorithms*: In this case, we attempt to partition the graph into groups of clusters so that each cluster contains groups of nodes which are densely connected. These densely connected groups of nodes may often provide significant information about how the entities in the underlying graph are interconnected with one another.
- *Graph clustering algorithms*: In this case, we have complete graphs available, and we wish to determine the clusters with the use of the structural information in the underlying graphs. Such cases are often encountered in the case of XML data, which are commonly encountered in many real domains.

We provided an overview of the different clustering algorithms available and the trade-offs with the use of different methods. The major challenges that remain in the area of graph clustering are as follows:

- *Clustering massive data sets*: In some cases, the data sets containing the graphs may be so large that they may be held only on disk. For example, if we have a dense graph containing 10^7 nodes, then the number of edges may be as high as 10^{13}. In such cases, it may not even be possible to store the graph effectively on disk. In the cases in which the graph can be stored on disk, it is critical that the algorithm should be designed in order to take the disk-resident behavior of the underlying data into account. This is especially challenging in the case of graph data sets, because the structural behavior of the graph interferes with our ability to process the edges sequentially for many applications. In the cases in which the graph is too large to store on disk, it is essential to design summary structures which can effectively store the underlying structural behavior of the graph. This stored summary can then be used effectively for graph clustering algorithms.

- *Clustering graph streams*: In this case, we have large graphs which are received as edge streams. Such graphs are more challenging, since a given edge cannot be processed more than once during the computation process. In such cases, summary structures need to be designed in order to facilitate an effective clustering process. These summary structures may be utilized in order to determine effective clusters in the underlying data. This approach is similar to the case discussed above in which the size of the graph is too large to store on disk.

In addition, techniques need to be designed for interfacing clustering algorithms with traditional database management techniques. In order to achieve this goal, effective representations and query languages need to be designed for graph data. This is a new and emerging area of research, and can be leveraged upon in order to further improve the effectiveness of graph algorithms.

Cross-References

▶ Group Detection
▶ Partitional Clustering

Recommended Reading

Abello J, Resende MG, Sudarsky S (2002) Massive quasi-clique detection. In: Proceedings of the 5th Latin American symposium on theoretical informatics (LATIN). Springer, Berlin, pp 598–612
Aggarwal C, Ta N, Feng J, Wang J, Zaki MJ (2007) XProj: a framework for projected structural clustering of XML documents. In: KDD conference, San Jose, pp 46–55
Ahuja R, Orlin J, Magnanti T (1992) Network flows: theory, algorithms, and applications. Prentice-Hall, Englewood Cliffs
Chawathe SS (1999) Comparing hierachical data in external memory. In: Very large data bases conference. Morgan Kaufmann, San Francisco, pp 90–101
Chung F (1997) Spectral graph theory. Conference Board of the Mathematical Sciences, Washington, DC
Dalamagas T, Cheng T, Winkel K, Sellis T (2005) Clustering XML documents using structural summaries. In: Information systems. Elsevier, Jan 2005
Gibson D, Kumar R, Tomkins A (2005) Discovering large dense subgraphs in massive graphs. In: VLDB conference, pp 721–732. http://www.vldb2005.org/program/paper/thu/p721-gibson.pdf
Jain A, Dubes R (1998) Algorithms for clustering data. Prentice-Hall, Englewood
Kernighan BW, Lin S (1970) An efficient heuristic procedure for partitioning graphs. Bell Syst Tech J 49:291–307
Lee M, Hsu W, Yang L, Yang X (2002) XClust: clustering XML schemas for effective integration. In: ACM conference on information and knowledge management. http://doi.acm.org/10.1145/584792.584841
Lian W, Cheung DW, Mamoulis N, Yiu S (2004) An efficient and scalable algorithm for clustering XML documents by structure. IEEE Trans Knowl Data Eng 16(1):82–96
Pei J, Jiang D, Zhang A (2005) On mining cross-graph quasi-cliques. In: ACM KDD conference, Chicago
Rattigan M, Maier M, Jensen D (2007) Graph clustering with network structure indices. In: Proceedings of the international conference on machine learning. ACM, New York, pp 783–790
Tsay AA, Lovejoy WS, Karger DR (1999) Random sampling in cut, flow, and network design problems. Math Oper Res 24(2):383–413
Zeng Z, Wang J, Zhou L, Karypis G (2007) Out-of-core coherent closed quasi-clique mining from large dense graph databases. ACM Trans Database Syst 32(2):13

Graph Kernels

Thomas Gärtner, Tamás Horváth, and
Stefan Wrobel
Fraunhofer IAIS, Schloss Birlinghoven,
University of Bonn, Sankt Augustin, Germany

Definition

The term *graph kernel* is used in two related but distinct contexts: On the one hand, graph kernels can be defined between graphs, that is, as a *kernel function* $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ where $\mathcal{G}$ denotes the set of all graphs un-der consideration. In the most common setting $\mathcal{G}$ is the set of all labeled undirected graphs. On the other hand, graph kernels can be defined between the vertices of a single graph, that is, as a kernel function $k : V \times V \to \mathbb{R}$ where V is the vertex set of the graph G under consideration. In the most common setting G is an undirected graph.

Motivation and Background

▶ Kernel methods are a class of machine learning algorithms that can be applied to any data set on which a valid, that is, positive definite, kernel function has been defined. Many kernel methods are theoretically well founded in statistical learning theory and have shown good predictive performance on many real–world learning problems.

Approaches for Kernels Between Graphs

One desireable property of kernels between graphs is that for non-isomorphic graphs $G, G' \in \mathcal{G}$ the functions $k(G, \cdot)$ and $k(G', \cdot)$ are not equivalent. If this property does not hold, the distance is only a pseudometric rather than a metric, that is, non-isomorphic graphs can be mapped to the same point in feature space and no kernel method can ever distinguish between the two graphs. However, it can be seen that computing graph kernels for which the property does hold is at least as hard as solving graph isomorphism (Gärtner et al. 2003).

For various classes of graphs, special purpose kernels have been defined such as for paths (▶ string kernels) and trees (Collins and Duffy 2002). These kernels are typically defined as the number of patterns that two objects have in common or as the inner product in a feature space counting the number of times a particular pattern occurs. The problem of computing a graph kernel where the patterns are all connected graphs, all cycles, or all paths and occurrence is determined by subgraph-isomorphism is, however, NP-hard (Gärtner et al. 2003).

Techniques that have been used to cope with the computational intractability of such graph kernels are (1) to restrict the considered patterns, for example, to bound the pattern size by a constant; (2) to restrict the class of graphs considered, for example, to trees or small graphs; (3) to define occurrence of a pattern differently, that is, not by subgraph-isomorphism; and (4) to approximate the graph kernel. Note that these four techniques can be combined.

While for technique (1) it is not immediately clear if the resulting graph kernel is feasible, technique (2) allows for fixed parameter tractable graph kernels. (Notice that even counting paths or cycles of length k in a graph is #W[1]-complete while the corresponding decision problem is fixed parameter tractable.) Though these will often still have prohibitive runtime requirements, it has been observed that enumerating cycles in real-world databases of small molecules is feasible (Horvath et al. 2004).

With respect to technique (3) it has been proposed to use graph kernels where the patterns are paths but the occurrences are determined by homomorphism (Gärtner et al. 2003; Kashima et al. 2003). Despite the explosion in the number of pattern occurrences (even very simple graphs can contain an infinite number of walks, that is, images of paths under homomorphism), if one downweights the influence of larger patterns appropriately, the kernel takes a finite value and closed form polynomial time computations exist. To increase the practical applicability of these graph kernels, it has been proposed to increase the number of labels by taking neighborhoods into account (Gärtner 2005) or to avoid "tottering" walks (Mahé et al. 2004).

Various approaches to approximate computation of graph kernels (4) exist. On the one hand, work on computing graph kernels based on restricting the patterns to frequent subgraphs (Deshpande et al. 2002) can be seen as approximations to the intractable all-subgraphs kernel. Computing such graph kernels is still NP-hard and no approximation guarantees are known. On the other hand, a recent graph kernel (Borgwardt et al. 2007) based on sampling small subgraphs of a graph at random is known to have a polynomial time algorithm with approximation guarantees.

The most common application scenario for such graph kernels is the prediction pharmaceutical activity of small molecules.

Approaches for Kernels on a Graph

Learning on the vertices of a graph is inherently *transductive*. Work on kernels between the vertices of a graph began with the "diffusion kernel" (Kondor and Lafferty 2002) and was later generalized in Smola and Kondor (2003) to a framework that contains the diffusion kernel as a special case. Intuitively, these kernels can be understood as comparing the neighborhoods of two vertices in the sense that the more neighbors two vertices have in common, the more similar they are. For classification, this definition is related to making the "cluster assumption", that is, assuming that the decision boundary between classes does not cross "high density" regions of the input space. To compute such graph kernels for increasing sizes of the neighborhood, one needs to compute the limit of a matrix poser series of the (normalized) graph Laplacian or its adjacency matrix. Different graph kernels arise from choosing different coefficients. In general, the limit of such matrix power series can be computed on the eigenvalues. For geometrically decaying parameters, the kernel matrix can also be computed by inverting a sparse matrix obtained by adding a small value to the diagonal of the Laplacian (in which case the kernel is called the "regularized Laplacian kernel") or the adjacency matrix.

In the case of the regularized Laplacian kernel, rather than first computing the kernel matrix and then applying an off-the-shelf implementation of a kernel method, it is often more effective to reformulate the optimization problem of the kernel method. Several possibilities for such reformulation have been proposed, including changing the variables as in Gärtner et al. (2006).

The most common application scenario for such graph kernels is the classification of entities in a social network.

Recommended Reading

Borgwardt KM, Petri T, Vishwanathan SVN, Kriegel H-P (2007) An efficient sampling scheme for comparison of large graphs. In: Mining and learning with graphs (MLG 2007), Firenze

Collins M, Duffy N (2002) Convolution kernel for natural language. In: Advances in neural information processing systems (NIPS), Vancouver, vol 16, pp 625–632

Deshpande M, Kuramochi M, Karypis G (2002) Automated approaches for classifying structures. In: Proceedings of the 2nd ACM SIGKDD workshop on data mining in bioinformatics (BIO KDD 2002), Edmonton

Gärtner T (2005) Predictive graph mining with kernel methods. In: Bandyopadhyay S, Maulik U, Holder LB, Cook DJ (eds) Advanced methods for knowledge discovery from complex data. Springer, Heidelberg, pp 95–121

Gärtner T, Flach PA, Wrobel S (2003) On graph kernels: hardness results and efficient alternatives. In: Proceedings of the 16th annual conference on computational learning theory and the 7th kernel workshop (COLT 2003), vol. 2777 of LNCS. Springer, Heidelberg, pp 129–143

Gärtner T, Le QV, Burton S, Smola AJ, Vishwanathan SVN (2006) Large-scale multiclass transduction. In: Advances in neural information processing systems, vol 18. MIT, Cambride, pp 411–418

Horvath T, Gärtner T, Wrobel S (2004) Cyclic pattern kernels for predictive graph mining. In: Proceedings of the international conference on knowledge discovery and data mining (KDD 2004). ACM, New York, pp 158–167

Kashima H, Tsuda K, Inokuchi A (2003) Marginalized kernels between labeled graphs. In: Proceedings of the 20th international conference on machine learning (ICML 2003). AAAI Press, Menlo Park, pp 321–328

Kondor RI, Lafferty J (2002) Diffusion kernels on graphs and other discrete input spaces. In: Sammut C, Hoffmann A (eds) Proceedings of the nineteenth international conference on machine learning (ICML 2002), pp. 315–322, Morgan Kaufmann, San Fransisco

Mahé P, Ueda N, Akutsu T, Perret J-L, Vert J-P (2004) Extensions of marginalized graph kernels. In: Proceedings of the 21st international conference on machine learning (ICML 2004). ACM, New York, p 70

Smola AJ, Kondor R (2003) Kernels and regularization on graphs. In: Proceedings of the 16th annual conference on computational learning theory and the 7th kernel workshop (COLT 2003). Volume 2777 of LNCS. Springer, Heidelberg, pp 144–158

Graph Mining

Deepayan Chakrabarti
Yahoo! Research, Sunnyvale, CA, USA

Definition

Graph Mining is the set of tools and techniques used to (a) analyze the properties of real-world graphs, (b) predict how the structure and properties of a given graph might affect some application, and (c) develop models that can generate realistic graphs that match the patterns found in real-world graphs of interest.

Motivation and Background

A graph $G = (V, E)$ consists of a set of edges, E connec-ting pairs of nodes from the set V; extensions allow for weights and labels on both nodes and edges. Graphs edges can be used to point *from* one node *to* another, in which case the graph is called directed; in an *undirected* graph, edges must point both ways: $i \rightarrow j \Leftrightarrow j \rightarrow i$. A variant is the bipartite graph $G = (V_1, V_2, E)$ where only edges linking nodes in V_1 to nodes in V_2 are allowed.

A graph provides a representation of the binary relationships between individual entities, and thus is an extremely common data structure. Examples include the graph of hyperlinks linking HTML documents on the Web, the social network graph of friendships between people, the bipartite graphs connecting users to the movies they like, and so on. As such, mining the graph can yield useful patterns (e.g., the communities in a social network) or help in applications (e.g., recommend new movies to a user based on movies liked by other "similar" users). Graph mining can also yield patterns that are common in many real-world graphs, which can then be used to design graph "generators" (e.g., a generator that simulates the Internet topology, for use in testing next-generation Internet protocols).

Structure of Learning System

We split up this discussion into three parts: the analysis of real-world graphs, realistic graph generators, and applications on graphs. Detailed surveys can be found in Newman (2003) and Chakrabarti and Faloutsos (2006).

Analysis of Real-World Graphs

Four basic types of large-scale patterns have been detected in real-world graphs. The first is the existence of power-laws, for instance in the degree distribution and eigenvalue distribution. Most nodes have very low degree while a few have huge degree. This has implications for algorithms whose running times are bounded by the highest degree. The second set of patterns is called the "small-world phenomenon," which state that the diameter (or effective diameter) of such graphs are very small with respect to their size. Recall that the diameter of a connected graph is the maximum number of hops needed to travel between any pair of nodes; the effective diameter is a more robust version that specifies the number of hops within which a large fraction (say, 90 %) of all pairs can reach each other. Examples include a diameter of around 4 for the Internet Autonomous System graph, around 19 for the entire US power grid, around 4 for the graph of actors who worked together in movies, and so on. Third, many large graphs exhibit "community effects," where each community consists of a set of nodes that are more tightly connected to other nodes in the community compared to nodes outside. One local manifestation of this effect is the relatively high *clustering coefficient* which counts, given all pairs of edges (i, j) and (j, k), the probability of the existence of the "transitive" edge (i, k). High clustering coefficients imply tight connections in neighborhoods, which is the basis of strong community structure. Finally, many large graphs were shown to increase in density as they evolve over time, that is, the number of edges grows according to a power-law on the number of nodes. In addition, even while more nodes and edges are being added, the diameter of the graph tends to decrease.

Graph Generators

Imagine designing an application that works on the Internet graph. Collecting the entire Internet graph in one place is hard, making the testing process for such an application infeasible. In such cases, a realistic graph generator can be used to simulate a large "Internet-like" graph, which can be used in place of the real graph. This synthetic graph must match the patterns typically found in the Internet, including the patterns discussed in the previous paragraph. Apart from generating such graphs, the generators can provide insights into the *process* by which large graphs came to attain their structure.

One example of this is the *preferential attachment* model. Starting with a small initial graph, this model adds one new node every step. The new node is connected to m previous nodes, with the probability of connecting to node i being proportional to its degree. This idea, popularly known as “the rich get richer,” can be shown to lead to a power-law degree distribution after a large number of nodes and edges have been added.

Many other models have also been proposed, which demonstrate graph generation as a random process, an optimization process, as a process on nodes embedded in some geographic space, and so on.

Applications

Some graph mining algorithms are meant to solve some application on any graph(s) provided as input to the algorithm. Several basic tools are commonly used in such applications, such as the ▸ Greedy Search Approach to Graph Mining the ▸ Inductive Database Search Approach to Graph Mining spectral methods, graph partitioning methods, and models based on random walks on graphs. *Tree Mining* is a special case of graph mining where the graphs are constrained to be trees. We will discuss a few such applications here.

Frequent subgraph mining: The aim is to find subgraphs that occur very frequently in the particular graph(s) in question (Kuramochi and Karypis 2001). This is quite useful in chemical datasets consisting of the graph structures of many different molecules (say, all protein molecules that have a certain chemical property); the frequent subgraphs in such molecules might represent basic structural units responsible for giving the molecules their special property. Unfortunately, the frequent subgraph problem subsumes the problem of subgraph isomorphism, and hence is NP-Hard. However, clever techniques have been devised to represent subgraphs so that checking for isomorphism can be done quickly in many cases.

Community detection: The problem is to detect tightly knit groups of nodes, where all nodes in the group have “similar” linkage structure. There are many algorithms, each optimizing for a different notion of similarity. Examples include graph partitioning methods such as spectral partitioning (Ng et al. 2002) and METIS that try to minimize the number of edges linking nodes across partitions, and co-clustering methods that aim for homogeneity in links across partitions.

Information diffusion and virus propagation: The spread of a contagious disease or a computer virus can be modeled (somewhat crudely) as a contact process on a graph, where the nodes are individuals who can get infected, and the links allow transmission of the contagion from an infected individual to an uninfected one. Similar models have been proposed to model the diffusion of information in social networks. The topology of the graph can be used to infer the most “influential” nodes in the graph, who are most capable of spreading the information quickly throughout the graph (Kempe et al. 2003).

Graph kernels: While subgraph isomorphism is a hard problem, we still need to be able to compare graphs on the basis of some similarity measure that can be computed in polynomial time. In the Kernel-Based Approach to Graph Mining graph kernels perform this task by computing similarities based on numbers of walks, paths, cyclic patterns, trees, etc.

Ranking on graphs: Given a graph (say, the Web hyperlink graph), we often need a ranking of the nodes in the graph. The ranking could be static (as in Page-Rank Brin and Page 1998) or it could depend on a user-specified query node. Such algorithms typically use some version of random walks on graphs (Lovász 1993), with the probability of the walk hitting a node being correlated with the importance of the node; such importances in turn yield a ranking of the nodes. Both static and query-dependent rankings can be useful in information retrieval settings, where a user desires information pertinent (i.e., “similar”) to her query.

Cross-References

- ▶ Graphs
- ▶ Graph Kernels
- ▶ Greedy Search Approach of Graph Mining
- ▶ Inductive Database Approach to Graphmining
- ▶ Link Mining and Link Discovery
- ▶ Tree Mining

Recommended Reading

Brin S, Page L (1998) The anatomy of a large-scale hypertextual web search engine. Comput Netw ISDN Syst 30(1–7):107–117

Chakrabarti D, Faloutsos C (2006) Graph mining: laws, generators and algorithms. ACM Comput Surv 38(1):2

Kempe D, Kleinberg J, Tardos E (2003) Maximizing the spread of influence through a social network. In: KDD, Washington, DC

Kuramochi M, Karypis G (2001) Frequent subgraph discovery. In: ICDM, San Jose, pp 313–320

Lovász L (1993) Random walks on graphs: a survey. In: Combinatorics: Paul Erdös is eighty, vol 2, pp 353–397

Newman MEJ (2003) The structure and function of complex networks. SIAM Rev 45:167–256

Ng A, Jordan M, Weiss Y (2002) On spectral clustering: analysis and an algorithm. In: NIPS, Vancouver

Graphical Models

Julian McAuley[1,2], Tibério Caetano[2], and Wray L. Buntine[2,3]
[1]Computer Science Department, University of California, San Diego, CA, USA
[2]Statistical Machine Learning Program, NICTA, Canberra, ACT, Australia
[3]Faculty of Information Technology, Monash University, Clayton, VIC, Australia

Definition

Graphical models are a means of compactly representing multivariate distributions, allowing for efficient algorithms to be developed when dealing with high-dimensional data. At their core, graphical models make use of the fact that high-dimensional distributions tend to factorize around local interactions, meaning that they can be expressed as a product of low-dimensional terms.

The notation we shall use is defined in Table 1, and some core definitions are presented in Table 2.

A few examples of the types of data that can be efficiently represented using graphical models are shown in Fig. 1. Here we have high-dimensional distributions (e.g., the probability of observing the pixels of a particular image), which we model in terms of low-dimensional interactions. In each of the examples presented in Fig. 1, we are simply asserting that

$$\underbrace{p(x_A, x_B | x_C)}_{\text{function of three variables}} = \underbrace{p(x_A | x_C)\, p(x_B | x_C)}_{\text{functions of two variables}}, \quad (1)$$

which arises by a straightforward application of the product rule (Definition 1), along with the fact that X_A and X_B are *conditionally independent*, given X_C (Definition 3). The key observation we make is that while the left-hand side of (Eq. 1) is a function of three variables, its conditional independence properties allow it to be *factored* into functions of two variables (note that the name "graphical models" arises due to the fact that such interdependencies can be represented as a graph encoding the relationships between variables).

In general, we shall have a series of conditional independence statements about X:

$$\left\{ X_{A_i} \perp X_{B_i} \;\middle|\; X_{C_i} \right\}. \quad (2)$$

It is precisely these statements that define the "structure" of our multivariate distribution, which we shall express in the form of a graphical model.

Motivation and Background

Graphical models are ubiquitous as a means to model multivariate data, since they allow us to represent high-dimensional distributions *compactly*; they do so by exploiting the *interdependencies* that typically exist in such

Graphical Models, Table 1 Notation

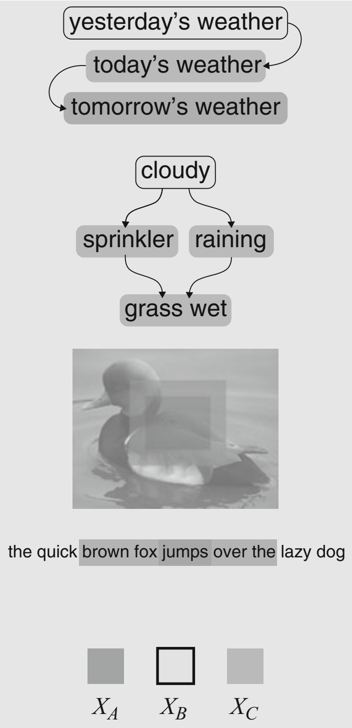
Notation	Description
$X = (X_1 \dots X_N)$	A random variable (we shall also use $X = (A, B, C \dots)$ in figures to improve readability)
$x = (x_1 \dots x_N)$	A *realization* of the random variable X
$\mathcal{X}$	The sample space (domain) of X
X_A	X can be indexed by a *set*, where we assume $A \subseteq \{1 \dots N\}$
$p(x)$	The probability that $X = x$
$\tilde{A}$	The *negation* of A, i.e., $\{1 \dots N\} \setminus A$
$X_A \perp X_B$	X_A and X_B are *independent*
$X_A \perp X_B \mid X_C$	X_A and X_B are *conditionally independent*, given X_C

Graphical Models, Table 2 Definitions

Definition 1 (product rule) $p(x_A, x_B) = p(x_A | x_B) p(x_B)$

Definition 2 (marginalization) $p(x_A) = \sum_{x_{\tilde{A}} \in \mathcal{X}_{\tilde{A}}} p(x_A, x_{\tilde{A}})$

Definition 3 (conditional independence) X_A and X_B are said to be conditionally independent (given X_C) iff $p(x_a | x_b, x_c) = p(x_a | x_c)$, for all x_a, x_b, and x_c; the conventional definition of "independence" is obtained by setting $X_C = \varnothing$



Graphical Models, Fig. 1 Some examples of conditional independence; we say that X_A and X_B are *conditionally independent*, given X_C, or more compactly $X_A \perp X_B \mid X_C$

data. Put simply, we can take advantage of the fact that high-dimensional distributions can often be decomposed into *low-dimensional factors* to develop efficient algorithms by making use of the distributive law: $ab + ac = a(b + c)$.

Some motivating examples are presented in Fig. 1; similar examples are ubiquitous in fields ranging from computer vision and pattern recognition to economics and the social sciences. Although we are dealing with high-dimensional data, we can make certain statements about the *structure* of the variables involved, allowing us to express important properties about the distribution compactly. Some of the properties we would like to compute include the probabilities of particular outcomes and the outcomes with the highest probability.

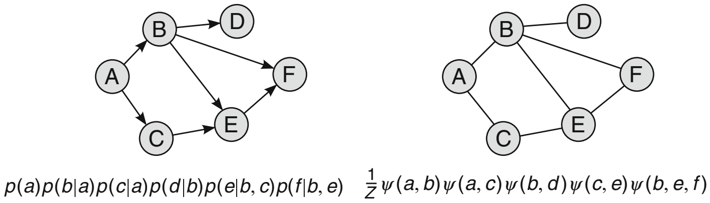
Theory

Directed Graphical Models

Due to the product rule (Definition 1), it is clear that *any* probability distribution can be written as

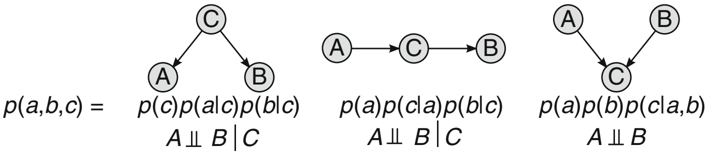
$$p(x) = \prod_{i=1}^{N} p(x_{\pi_i} | x_{<\pi_i}) \qquad (3)$$

for an arbitrary permutation π of the labels, where we define $<i := \{1 \dots i - 1\}$. For example any four-dimensional distribution can be written



B
D
A
F
C
E
$p(a)p(b|a)p(c|a)p(d|b)p(e|b,c)p(f|b,e)$
B
D
A
F
C
E
$\frac{1}{Z}\psi(a,b)\psi(a,c)\psi(b,d)\psi(c,e)\psi(b,e,f)$

Graphical Models, Fig. 2 A directed model (*left*) and an undirected model (*right*). The joint distributions they represent are shown



Graphical Models, Fig. 3 Some simple Bayesian Networks and their implied independence statements. Note in particular that in the rightmost example, we *do not* have $A \perp B \mid C$

as

$$p(x_a, x_b, x_c, x_d) = p(x_c)p(x_b|x_c)p(x_d|x_c, x_b) \\ p(x_a|x_c, x_b, x_d). \qquad (4)$$

With this idea in mind, consider a model $p(x)$ for which we have the conditional independence statements:

$$\left\{p(x_{\pi_i}|x_{<\pi_i}) = p(x_{\pi_i}|x_{pa_{\pi_i}})\right\}, \qquad (5)$$

where $pa_{\pi_i} \subset <\pi_i$. We now have

$$p(x) = \prod_{i=1}^{N} p(x_{\pi_i}|x_{pa_{\pi_i}}). \qquad (6)$$

We can interpret pa_i as referring to the "parents" of the node i. Essentially, we are saying that a variable is conditionally independent on its nondescendants, given its parents.

We can represent (Eq. 6) using a directed acyclic graph (DAG) by representing each variable X_i as a node; an arrow is formed from X_j to X_i if $j \in pa_i$. An example of such a representation is given in Fig. 2. It can easily be shown that the resulting graph is always acyclic.

A *Bayesian Network* (a type of *directed* graphical model) is simply a set of probability distributions of the form $p(x) = \prod_{i=1}^{N} p(x_i|x_{pa_i})$. Every Bayesian Network can be represented as a DAG, though we often simply say that the Bayesian Network "is" the DAG. Some trivial examples and the type of independence statements they imply are shown in Fig. 3.

We finish this section with a simple lemma:

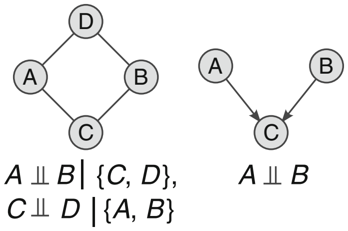
Lemma 1 (Topological Sort) *Every DAG has at least one permutation π that "sorts" the nodes such that each node has a larger index than its parents; in other words, the factorization associated to any DAG can be written in the form of (Eq. 6) for at least one π such that $\pi_i > j$ for all i, where $j \in pa_{\pi_i}$.*

Undirected Graphical Models

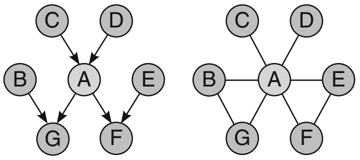
Although we have shown how conditional independence statements in the form of (Eq. 5) can be modeled using a DAG, there exist certain conditional independence statements that are not satisfied by *any* Bayesian Network, such as those in Fig. 4.

Markov random fields (or MRFs) allow for the specification of a different class of conditional independence statements, which are naturally represented by *undirected graphs* (UGs for short). The results associated with MRFs require a few additional definitions:

Definition 4 (Clique) A set of nodes X in a graph $\mathcal{G} = (V, E)$ is said to form a clique if $(X_i, X_j) \in E$ for every $X_i, X_j \in X$ (i.e., the subgraph X is fully connected).


D
A
B
C
$A \perp\!\!\!\perp B \mid \{C, D\}$,
$C \perp\!\!\!\perp D \mid \{A, B\}$
A
B
C
$A \perp\!\!\!\perp B$

Graphical Models, Fig. 4 There is no Bayesian Network that captures precisely the conditional independence properties of the Markov random field at *left*; there is no Markov random field that captures precisely the conditional independence properties of the Bayesian Network at *right*



Graphical Models, Fig. 5 The Markov blanket of the node A consists of its parents, its children, and the parents of its children (*left*). The corresponding structure for *undirected* models simply consists of the neighbors of A. Note that if we convert the directed model to an undirected one (using the procedure described in section "Conversion from Directed to Undirected Graphical Models"), then the Markov blankets of the two graphs are identical

Definition 5 (Maximal Clique) A clique X is said to be maximal if there is no clique Y such that $X \subset Y$.

A Markov random field is a probability distribution of the form $p(x) = \frac{1}{Z}\prod_{c \in \mathcal{C}} \psi_c(x_c)$, where $\mathcal{C}$ is the set of maximal cliques of $\mathcal{G}$, ψ_c is an arbitrary nonnegative real-valued function, and Z is simply a normalization constant ensuring that $\sum_x p(x) = 1$.

Conversion from Directed to Undirected Graphical Models

It is possible to convert a directed graphical model to an undirected graphical model via the following simple procedure:

- For every node X_i with parents pa_{X_i}, add undirected edges between every $X_j, X_k \in pa_{X_i}$.
- Replace all directed edges with undirected edges.

In other words, we are replacing statements of the form $p(x_A|x_B)$ with $\psi(x_A, x_B)$, so that the nodes $\{X_i\} \cup pa_{X_i}$ now form a clique in the undirected model. This procedure of "marrying the parents" is referred to as *moralization*. Naturally, the undirected model formed by this procedure does not precisely capture the conditional independence relationships in the directed version. For example, if it is applied to the graph in Fig. 4 (*right*), then the nodes A, B, and C form a clique in the resulting model, which does not capture the fact that $A \perp B$. However, we note that every term of the form $p(x_i|x_{pa_i})$ appears in some clique of the undirected model, meaning that it can include all of the factors implied by the Bayesian Network.

Characterization of Directed and Undirected Graphical Models

We can now present some theorems that characterize both Bayesian Networks and Markov random fields:

Lemma 2 (Local Markov Property) *A node in a DAG is conditionally independent of its non-descendants, given its parents (this is referred to as the "directed" local Markov property); a node in a UG is conditionally independent of its non-neighbors, given its neighbors.*

Definition 6 (Markov Blanket) Given a node A, its "Markov blanket" is the minimal set of nodes C such that $A \perp B \mid C$ for all other nodes B in the model (in other words, the minimal set of nodes that we must know to "predict" the behavior of A).

Lemma 3 (Markov Blankets of Directed and Undirected Graphs) *In a directed network, the Markov blanket of a node A (denoted $MB(A)$) consists of its parents, its children, and its children's (other) parents. In an undirected network, it simply consists of the node's neighbors (see Fig. 5).*

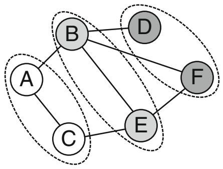
Definition 7 (d-separation) The notion of a Markov blanket can be generalized to the notion of "d-separation." A set of nodes A is said to be d-separated from a set B by a set C if every (undirected) path between A and B is "blocked" when C is in the conditioning set (i.e., when C is observed). A path is said to be blocked if **either** it contains (p_1, p_2, p_3) with $p_1 \rightarrow p_2 \leftarrow p_3$ (where arrows indicate edge directions) and neither p_2 nor any of its descendants are observed, **or** it contains (p_1, p_2, p_3) with $p_1 \rightarrow p_2 \rightarrow p_3$ and p_2 is observed, **or** it contains (p_1, p_2, p_3) with $p_1 \leftarrow p_2 \rightarrow p_3$ and p_2 is observed.

Applying (Definition 7) to the directed graphs in Fig. 1, we would say that the aqua regions (X_C) *d-separate* the red regions (X_A) from the white regions (X_B); *all conditional independence statements can simply be interpreted as d-separation in a DAG.*

The analogous notion of *graph separation* for Markov random fields is simpler than that of d-separation for Bayesian Networks. Given an undirected graph $\mathcal{G}$ and disjoint subsets of nodes A, B, C, if A is only reachable from B via C, this means that A is *separated* from B by C and these semantics encode the probabilistic fact that $A \perp B \mid C$. This is illustrated in Fig. 6.

In both the directed and undirected case, a Markov blanket of a node is simply the minimal set of nodes that d-separates/graph separates that node from all others.

A complete characterization of the class of probability distributions represented by Bayesian Networks can be obtained naturally once conditional independence statements are mapped to d-separation statements in a DAG. The following theorem settles this characterization.



Graphical Models, Fig. 6 The nodes $\{B, E\}$ form a *clique*; the nodes $\{B, E, F\}$ form a *maximal clique*. The nodes $\{B, E\}$ *separate* the nodes $\{A, C\}$ from $\{D, F\}$

Theorem 1 *Let p be a probability distribution that satisfies the conditional independence statements implied by d-separation in a DAG. Then p factors according to (Eq. 6). The converse also holds.*

For Markov random fields, an analogous characterization exists:

Theorem 2 (Hammersley-Clifford) *If a strictly positive probability distribution p satisfies the conditional independence statements implied by graph separation in an undirected graph $\mathcal{G}$, then*

$$p(x) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \psi_c(x_c). \tag{7}$$

The converse also holds, albeit in a more general sense in that p need not be strictly positive.

It can be shown that

directed local Markov property		*local Markov property*
⇕		⇕
d-separation in a DAG	and (for positive p) that	*graph separation in a UG*
⇕		⇕
factorization of p by (Eq. 6)		*factorization of p by* (Eq. 7)

Knowing that directed models can be converted to undirected models, we shall consider inference algorithms in undirected models only.

Applications

Inference Algorithms in Graphical Models

The key observation that we shall rely on in order to do inference efficiently is the *distributive law*:

$$\underbrace{ab + ac}_{\text{three operations}} = \underbrace{a(b + c)}_{\text{two operations}}. \tag{8}$$

By exploiting the factorization in a graphical model, we can use this law to perform certain queries efficiently (such as computing the marginal with respect to a certain variable).

As an example, suppose we wish to compute the marginal $p(x_1)$ in an MRF with the following factorization:

$$p(x) = \frac{1}{Z} \prod_{i=1}^{N-1} \psi(x_i, x_{i+1}). \tag{9}$$

Note that the graph representing this model is simply a *chain*. Computing the sum in the naïve way requires computing

$$p(x_1) = \frac{1}{Z} \sum_{x_{\{2 \dots N\}}} \prod_{i=1}^{N-1} \psi(x_i, x_{i+1}), \tag{10}$$

whose complexity is $\Theta(\prod_{i=1}^{N} |\mathcal{X}_i|)$. However, due to the distributive law, the same result is simply

$$\begin{aligned} p(x_1) = \frac{1}{Z} \sum_{x_2} \Big[\psi(x_1, x_2) \sum_{x_3} \Big[\psi(x_2, x_3) \cdots \\ \sum_{x_{N-1}} \Big[\psi(x_{N-2}, x_{N-1}) \\ \sum_{x_N} \psi(x_{N-1}, x_N) \Big]\Big]\Big], \end{aligned} \tag{11}$$

whose complexity is $\Theta(\sum_{i=1}^{N-1} |\mathcal{X}_i||\mathcal{X}_{i+1}|)$. As a more involved example, consider computing the marginal with respect to A in the undirected model in Fig. 2; here we wish to compute

$$p(a) = \frac{1}{Z} \sum_{b,c,d,e,f} \psi(a,b)\psi(a,c)\psi(b,d) \psi(c,e)\psi(b,e,f) \tag{12}$$

$$= \frac{1}{Z} \sum_{b} \psi(a,b) \sum_{c} \psi(a,c) \sum_{d} \psi(b,d) \sum_{e} \psi(c,e) \sum_{f} \psi(b,e,f). \tag{13}$$

Exploiting the distributive law in this way is often referred to as the *Elimination Algorithm*. It is useful for computing the marginal with respect to a single variable. However, should we wish to compute the marginal with respect to *each* variable, for example, it is not an efficient algorithm as several operations shall be repeated.

Belief Propagation

In tree-structured models, the elimination algorithm can be adapted to avoid repeated computations, using a message-passing scheme known as *belief propagation*, or the *sum-product* algorithm. This is presented in Algorithm 3. Here the "cliques" in the model are simply edges. This algorithm was invented independently by many authors and is the most efficient among many variations.

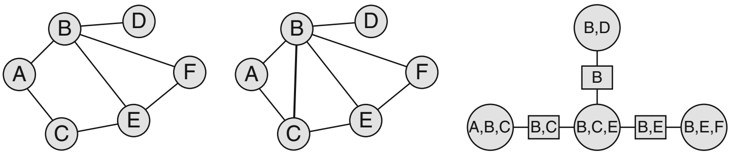
It can be easily demonstrated that the condition in Algorithm 3, Line 3, is always satisfied by some pair of edges until all messages have been passed: initially, it is satisfied by all of the "leaves" of the model; messages are then propagated inward until they reach the "root" of the tree; they are then propagated outward.

Maximum a Posteriori (MAP) Estimation

Algorithm 3 allows us to compute the *marginals* of the variables in a graphical model. There are other related properties that we may also wish to compute, such as finding which states have the

Algorithm 3 The *sum-product* algorithm

```
Input: an undirected, tree-structured graphical model $\mathcal{X}$
    with cliques $\mathcal{C}$ {the cliques are simply edges in this case}
1: define $m_{A \to B}(x_{A \cap B})$ to be the "message" from an edge $A$ to an adjacent edge $B$ {for example if $A = (a,b)$ and $B = (b,c)$ then we have $m_{(a,b) \to (b,c)}(x_b)$}
2: while there exist adjacent edges $A, B \in \mathcal{C}$ for which $m_{A \to B}$ has not been computed do
3:     find some $A \in \mathcal{C}$ such that $m_{C \to A}$ has been computed for every neighbor $C \in \Gamma(A)$, except $B$ {$\Gamma(A)$ returns the edges neighboring $A$; initially the condition is satisfied by all leaf-edges}
4:     $m_{A \to B}(x_{A \cap B}) := \sum_{x_{A \setminus B}} \{\psi_A(x_A) \prod_{C \in \Gamma(A) \setminus B} m_{C \to A}(x_{A \cap C})\}$
5: end while
6: for $A \in \mathcal{C}'$ do
7:     $marginal_A(x_A) := \psi_A(x_A) \prod_{C \in \Gamma(A)} m_{C \to A}(x_{A \cap C})$
8: end for
```



Graphical Models, Fig. 7 The graph at *left* is not chordal, since the cycle (A, B, E, C) does not contain a chord; adding the edge (B, C) results in a chordal (or triangulated) graph (*center*). The graph at *right* is a junction tree for the graph at *center*; the cliques of the triangulated graph form the nodes (*circles*); their *intersection sets* are shown as squares. Note that this is not the only junction tree that we could form – the node $\{B, D\}$ could connect to any of the other three nodes

highest probability (the *maximum a posteriori*, or simply "MAP" states). To do so, we note that the operations $(+, \times)$ used in Algorithm 3 can be replaced by $(\max, \times)$. This variant is usually referred to as the *max-product* (as opposed to *sum-product*) algorithm. Indeed, different quantities can be computed by replacing $(+, \times)$ by any pair of operations that form a *semiring* (Aji and McEliece 2000).

The Junction Tree Algorithm

Algorithm 3 applies only for tree-structured graphs. We can generalize this algorithm to general graphs. We do so by working with a different type of tree-structured graph, whose *nodes* contain the *cliques* in our original graph. We begin with some definitions:

Definition 8 (Chordal Graph) A graph $\mathcal{G}$ is said to be chordal if every cycle $(c_1 \dots c_n)$ in $\mathcal{G}$ contains a chord (i.e., an edge (c_i, c_j) such that $j > (i + 1)$).

Definition 9 (Clique Graph, Clique Tree) A clique graph $\mathcal{H}$ of a graph $\mathcal{G}$ is a graph whose nodes consist of (maximal) cliques in $\mathcal{G}$ and whose edges correspond to intersecting cliques in $\mathcal{G}$. A clique tree is a clique graph without cycles.

Definition 10 (Junction Tree) A clique tree $\mathcal{H}$ of $\mathcal{G}$ is said to form a junction tree if for every pair of nodes A, B (i.e., maximal cliques in $\mathcal{G}$), the path between them $(P_1 \dots P_m)$ satisfies $(A \cap B) \subset P_i$ for all $i \in \{1 \dots m\}$.

The algorithms we shall define apply only if the graph in question is *chordal*, or "triangulated" (Definition 8); this can always be achieved by adding additional edges to the graph, as demonstrated in Fig. 7; adding additional edges means increasing the size of the maximal cliques in the graph.

Finding the "optimal" triangulation (i.e., the one that minimizes the size of the maximal cliques) is an NP-complete problem. In practice, triangulation algorithms vary from simple greedy heuristics (e.g., select a node that has as few neighbors as possible) to complex approximation algorithms working within a factor of the optimal solution (Amir 2001).

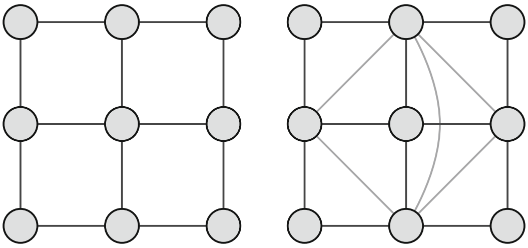
The problem of actually *generating* a junction tree from the triangulated graph is easily solved by a maximum spanning tree algorithm (where we prefer edges corresponding to pairs of cliques with large intersections).

Theorem 3 *Let $\mathcal{G}$ be a triangulated graph and $\mathcal{H}$ a corresponding clique tree. If the sum of the cardinalities of the intersection sets of $\mathcal{H}$ is maximum, then $\mathcal{H}$ is a junction tree. The converse also holds.*

If the nodes and edges in Algorithm 3 are replaced by the nodes (maximal cliques in $\mathcal{G}$) and edges (intersecting cliques in $\mathcal{G}$) of $\mathcal{H}$, then we recover the *junction tree algorithm*.

Approximate Inference

The act of triangulating the graph in the junction tree algorithm may have the effect of increasing the size of its maximal cliques, as in Fig. 8. This may be a problem, as its running time is exponential in the size of the maximal cliques in the *triangulated* graph (this size minus one is referred to as the *tree-width* of the graph, e.g., a chain has a tree-width of 1).



Graphical Models, Fig. 8 The graph above at *left* has maximal cliques of size two; in order to triangulate it, we must introduce maximal cliques of size four (*right*)

There are a variety of approximate algorithms that allow us to perform inference more efficiently:

Variational approximation. If doing inference in a graphical model $\mathcal{X}$ is intractable, we might search for a model $\mathcal{Y}$ for which inference is tractable and which is "similar" to $\mathcal{X}$ in terms of the KL-divergence between $p(x)$ and $p(y)$ (Wainwright and Jordan 2008).

Loopy belief propagation. We can build a clique graph from a graph that has not been triangulated, simply by connecting all cliques that intersect (in which case, the clique graph will contain loops). If we then propagate messages in some *random* order, we can obtain good approximations under certain conditions (Ihler et al. 2005).

Gibbs sampling. Given an estimate $x_{A\backslash B}$ of a set of variables $X_{A\backslash B}$, we can obtain an estimate of x_B by sampling from the conditional distribution $p(x_B|x_{A\backslash B})$. If we choose $B = \{X_i\}$, and repeat the procedure for random choices of $i \in \{1 \ldots N\}$, we obtain the procedure known as *Gibbs sampling* (Geman and Geman 1984).

There are several excellent books and tutorial papers on graphical models. A selection of tutorial papers includes Aji and McEliece (2000), Kschischang et al. (2001), Murphy (1998), and Wainwright and Jordan (2008); review articles include Roweis and Ghahramani (1997) and Smyth (1998), to name but a few.

Other signicant works include Koller and Friedman (2009), Jensen (2001) (introductory books), Edwards (2000) (undirected models), Pearl (1988, 2000) (directed models), Cowell et al. (2003) (exact inference), Jordan (1998) (learning and approximate inference), and Lauritzen (1996, Lauritzen and Spiegelhalter 1988) (a comprehensive mathematical theory).

There is also a variety of closely related models and extensions:

Gaussian graphical models. We have assumed throughout that our probability distributions are *discrete*; however, the only condition we require is that they are *closed under multiplication and marginalization*. This property is also satisfied for *Gaussian* random variables.

Hidden Markov models. In many applications, the variables in our model may be *hidden*. The above algorithms can be adapted to infer properties about our hidden states, given a sequence of observations.

Kalman filters. Kalman filters employ both of the above ideas, in that they include hidden state variables taking values from a *continuous* space using a Gaussian noise model. They are used to estimate the states of *linear dynamic systems* under noise.

Factor graphs. Factor graphs employ an alternate message-passing scheme, which may be preferable for computational reasons. Inference remains approximate in graphs with loops, though approximate solutions may be obtained more efficiently than by loopy belief propagation (Kschischang et al. 2001).

Relational models. Relational models allow us to explore the relationships between objects in order to predict the behavior and properties of each. Graphical models are used to predict the properties of an object based on others that relate to it (Getoor and Taskar 2007).

Learning. Often, we would like to *learn* either the parameters or the structure of the model from (possibly incomplete) data. There is an extensive variety of approaches; a collection of papers appears in Jordan (1998).

Deep learning. Deep belief networks can also be viewed as instances of graphical models, which impose a particular structure on the relationships between input variables, output variables, and hidden units. In particular, deep belief nets assume that complex relationships can be broken down into (a massive number of) purely pairwise interactions.

Cross-References

- ▶ Bayesian Network
- ▶ Deep Belief Nets
- ▶ Expectation Propagation
- ▶ Hidden Markov Models
- ▶ Markov Random Field

Recommended Reading

Aji SM, McEliece RJ (2000) The generalized distributive law. IEEE Trans Inf Theory 46(2):325–343

Amir E (2001) Efficient approximation for triangulation of minimum treewidth. In: Proceedings of the 17th conference on uncertainty in artificial intelligence. Morgan Kaufmann, San Francisco, pp 7–15

Cowell RG, Dawid PA, Lauritzen SL, Spiegelhalter DJ (2003) Probabilistic networks and expert systems. Springer, Berlin

Edwards D (2000) Introduction to graphical modelling. Springer, New York

Geman S, Geman D (1984) Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. IEEE Trans Pattern Anal Mach Intell 6:721–741

Getoor L, Taskar B (eds) (2007) An introduction to statistical relational learning. MIT Press, Cambridge

Ihler AT, Fischer III JW, Willsky AS (2005) Loopy belief propagation: convergence and effects of message errors. J Mach Learn Res 6:905–936

Jensen FV (2001) Bayesian networks and decision graphs. Springer, Berlin

Jordan M (ed) (1998) Learning in graphical models. MIT Press, Cambridge

Koller D, Friedman N (2009) Probabilistic graphical models: principles and techniques. MIT Press, Cambridge

Kschischang FR, Frey BJ, Loeliger HA (2001) Factor graphs and the sum-product algorithm. IEEE Trans Inf Theory 47(2):498–519

Lauritzen SL (1996) Graphical models. Oxford University Press, Oxford

Lauritzen SL, Spiegelhalter DJ (1988) Local computations with probabilities on graphical structures and their application to expert systems. J R Stat Soc Ser B 50:157–224

Murphy K (1998) A brief introduction to graphical models and Bayesian networks. Morgan Kaufmann, San Francisco

Pearl J (1988) Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann, San Francisco

Pearl J (2000) Causality. Cambridge University Press, Cambridge

Roweis S, Ghahramani Z (1997) A unifying review of linear Gaussian models. Neural Comput 11:305–345

Smyth P (1998) Belief networks, hidden Markov models, and Markov random fields: a unifying view. Pattern Recogn Lett 18:1261–1268

Wainwright MJ, Jordan MI (2008) Graphical models, exponential families, and variational inference. Found Trends Mach Learn 1:1–305

Graphs

Tommy R. Jensen
Alpen-Adria-Universität Klagenfurt, Klagenfurt, Austria

Definition

Graph Theory is (dyadic) relations on collections specified objects. In its most common, a *graph* is a pair $G = (V, E)$ of a (finite) set of *vertices* V and a set of *edges* E (or *links*). Each edge e is a 2-element subset $\{u, v\}$ of V, usually abbreviated as $e = uv$; u and v are called the *endvertices* of e, they are mutually *adjacent* and each is *incident* to e in G. This explains the typical model of a *simple graph*.

A *directed graph* or ▶ digraph is a more general structure, in which the edges are replaced by ordered pairs of distinct elements of the vertex set V, each such pair being referred to as an *arc*. Another generalization of a graph is a *hypergraph* or “set-system” on V, in which the *hyperedges* may have any size. Various concepts in graph theory extend naturally to *multigraphs*, in which each pair of (possibly identical) vertices may be adjacent via several edges (respectively *loops*). Also studied are *infinite graphs*, for which the vertex and edge sets are not restricted to be finite.

A graph is conveniently depicted graphically by representing each vertex as a small circle, and representing each edge by a curve that joins its two endvertices. A digraph is similarly depicted by adding an arrow on the curve representing an *arc* showing the direction from its *tail* to its (possibly identical) *head*.

Motivation and Background

One of the very first results in graph theory appeared in Leonhard Euler's paper on Seven Bridges of Königsberg, published in 1736. The paper contained the complete solution to the problem whether, when given a graph, it is possible to locate an *Euler tour*, that is, a sequence of adjacent edges (each edge imagined to be traversed from one end to the other) that uses every edge exactly once. Figure 1 illustrates the four main parts of the city of Königsberg with the seven bridges connecting them; since this graph contains four vertices of odd degree, it does not allow an Euler tour.

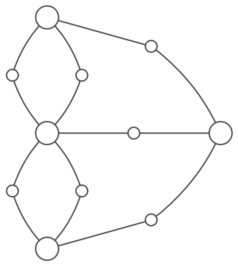
Applications of graphs are numerous and widespread. Much of the success of graph theory is due to the ease at which ideas and proofs may be communicated pictorially in place of, or in conjunction with, the use of purely formal symbolism.

Theory

Isomorphism

A graph drawing should not be confused with the graph itself (the underlying abstract structure) as there are several ways to structure the graph drawing. It only matters which vertices are connected to which others by how many edges, the exact layout may be suited for the particular purpose at hand. It is often a problem of independent interest to optimize a drawing of a given graph in terms of aesthetic features.

Graphs, Fig. 1 A graph of the city of Königsberg

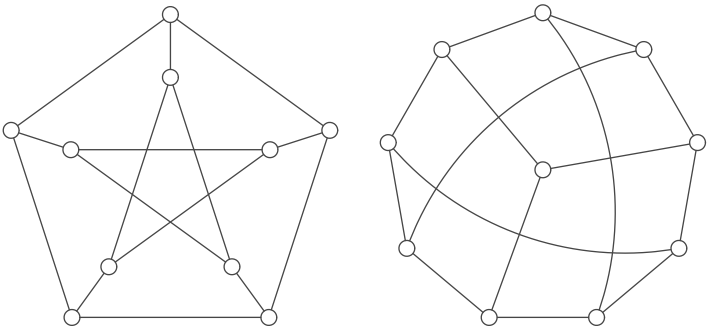


In practice it is often difficult to decide if two drawings represent the same graph (as in Fig. 2). This decision problem has gained increasing status in complexity theory, with growing suspicion that this problem may fall in a new class of problems, which lies between the familar classes of polynomially solvable and NP-complete ► (NP-completeness) problems (supposing that these classes are indeed distinct; for issues related to the complexities of decision and optimization problems see Garey and Johnson 1979). Nonetheless it is customary in the treatment of abstract graphs to consider two graphs identical if they are isomorphic. A closely related problem, the *subgraph isomorphism problem*, an NP-complete problem, consists in finding a given graph as a subgraph of another given graph.

Whereas there seems common agreement in the graph theoretic community on what constitutes a drawing of a graph, it may be considered a weakness, and sometimes a source of confusion, that even the most central general sources on the fundamentals of graph theory, such as the monographs (Berge 1976; Bondy and Murty 2007; Diestel 2005), do not agree on a common formalization of the theory.

Classes of Graphs

Important special classes of graphs are *bipartite graphs*, for which the vertex set is partitionable into two classes A, B with every edge having one end in A and one in B; in particular the *complete bipartite graph* $K_{m,n}$ has $|A| = m, |B| = n$, and every vertex in A is joined to every vertex in B. The *complete graph* K_n consists of n vertices that are all pairwise adjacent. A *path* of length n consists of vertices $v_0, v_1, \ldots, v_n$ with edges $v_{i-1}v_i$ for $i = 1, 2, \ldots, n$; such a path *joins* its two endvertices v_0 and v_n. A *circuit* of length n consists of a path of length $n - 1$ together with an additional edge between the two endvertices of the path. A graph is *connected* if each pair of its vertices is joined by at least one path within the graph. Of central importance to the study of efficient search procedures in computer science is the class of *trees*, those connected graphs that



Graphs, Fig. 2 Two drawings of the same graph

contain no circuits. Most definitions have various natural counterparts for directed graphs, in particular a *tournament* is a directed graph in which each pair of vertices is joined by exactly one arc.

Properties of Graphs

Finding a complete subgraph of a given order in an input graph is called the *clique problem*. The complementary problem of finding an independent set is called the *independent set problem*. The *longest path problem* and the *longest circuit problem* have as special cases the *Hamilton path problem* and the *Hamilton circuit problem*, the latter two problems asking to find a path, respectively a circuit, that uses all vertices of the given graph. Each of these problems (or a suitable modification of it) belongs to the complexity class of NP-complete problems, hence is generally believed to be very difficult to solve efficiently. The weighted version of the Hamilton circuit problem, the so-called *travelling salesman problem* is of central importance in combinatorial optimization.

A graph is called *planar* if it may be drawn in the Euclidian plane without any two of its edges crossing except where they meet at a common endvertex. This is often a convenient way of representing a graph, whenever it is doable. A theorem of Kuratowski states that a graph is planar if and only if it contains homeomorphic copies of neither the complete bipartite graph $K_{3,3}$ (the three-houses-three-utilities-graph) nor the complete graph K_5. A main branch of graph theory is concerned with investigating relationships between the topological and combinatorial properties of graphs (Mohar and Thomassen 2001).

In 1852, Francis Guthrie posed the *four color problem*, asking if it is possible to color the countries of any map, using only four colors, in such a way that all pairs of bordering countries receive different colors. Equivalently, by representing dually every country as a vertex of a graph, with two vertices joined by an edge if their countries share a stretch of common border, the question is whether it is possible to color the vertices of a planar graph using four colors, so that any two adjacent vertices receive distinct colors. This problem, was solved a century later in 1976 by Kenneth Appel, Wolfgang Haken, and John Koch, who invested massive amounts of computing time to complete a graph theoretic approach developed by various mathematicians over a period of most of the preceding part of the twentieth century.

The problem of *coloring* a possibly nonplanar graph with a minimal number of colors, that is, to partition its vertex set into as few independent sets as possible, is a well-studied problem (e.g., see Jensen and Toft 1995), though NP-hard in general. In fact it is already an NP-complete problem to ask whether a given planar graph allows a coloring using at most three colors (see Garey et al. 1976). The recent strong perfect graph theorem provides one of quite few known examples of a fairly rich class of graphs, the *Berge graphs*, for which the coloring problem has a satisfactory solution (see Chudnovsky et al. 2006).

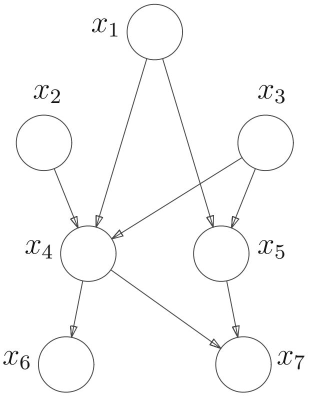
Other well-solved problems include finding a largest *matching* in a given graph; a largest set of edges no two of which share a common endvertex (see Lovász and Plummer (1986) for a thorough treatment of *matching theory*). The most interesting special case asks to find a *perfect matching*, having the property that every vertex is paired up with a unique vertex of the graph adjacent to it. For the special case of bipartite graphs (the *marriage problem*), the problem was solved by Dénes König in 1931. Even when given for every pair of vertices a measure of the desirability of pairing up these particular vertices (the *weighted matching problem*), there exists an efficient solution to the problem of finding an optimum matching of maximal total weight, discovered by Jack Edmonds in 1959.

Applications

As an example of a visualization application, Fig. 3 shows a digraph to symbolize for a collection of seven stochastic variables $x_1, \ldots, x_7$ that their joint distribution is given by the product

$$p(x_1)\,p(x_2)\,p(x_3)\,p(x_4|x_1,x_2,x_3)\,p(x_5|x_1,x_3) \\ \times\, p(x_6|x_4)\,p(x_7|x_4,x_5) \qquad (1)$$

In addition to visualization of a network, a process, a search procedure, or any hierarchical structure, there are many applications using implementations of known graph algorithms on computers, so that the graph in question will only exist as an abstract datastructure within a program and thus remains invisible to the user.



Graphs, Fig. 3 Reproduced from Bishop (2006, p. 362)

There are different ways to store graphs in a computer. Often a combination of list and matrix structures will be preferred for storage and dynamic manipulation of a graph by an algorithm. List structures are often preferred for sparse graphs as they have smaller memory requirements. Matrix structures on the other hand provide faster access but can consume a large amount of memory if a graph contains many vertices. In most cases it is convenient to represent a graph or digraph by an array containing, for each edge or arc, the pair of vertices that it joins, together with additional information, such as the weight of the edge, as appropriate. It may be an advantage in addition to store for each vertex a list of the vertices adjacent to it, or alternatively, a list of the edges incident to it, depending on the application.

The adjacency matrix of a graph, multigraph, or digraph on n vertices is an $n \times n$ matrix in which the *ij*-entry is the number of edges or arcs that join vertex i to vertex j (or more generally, the weight of a single such edge or arc). As a storage device this is inferior for sparse graphs, those with relatively few edges, but gains in importance when an application naturally deals with very dense graphs or multigraphs.

Future Directions

In recent years the theory of *graph minors* has been an important focus of graph theoretic research. A graph H is said to be a *minor* of a graph G if there exists a subgraph of G from which H can be obtained through a sequence of *edge contractions*, each consisting of the identification of the two ends of an edge e followed by the removal of e. A monumental effort by Neil Robertson and Paul Seymour has resulted in a proof of the Robertson–Seymour theorem (Robertson and Seymour 2004; see also Diestel 2005), with the important consequence that for any set $\mathcal{G}$ of graphs that is closed under taking minors, there exists a finite set of obstruction graphs, such that G is an element of $\mathcal{G}$ precisely if G does not contain any minor that belongs to the obstruction set. This theorem has several important algorithmic consequences, many still waiting to be fully explored.

A particularly challenging unsolved problem is the *Hadwiger conjecture* (see Jensen and Toft 1995), stating that any graph G that does not allow a vertex coloring with as few as k colors will have to contain the complete graph K_{k+1} as a minor. The special cases of $k \leq 5$ colors have been shown to be consequences of the four color theorem. But the problem remains open for all larger values of k.

Other central areas of research relate to the notoriously hard problems of vertex- and edge-coloring, and of Hamilton paths and circuits. These have important applications, but it is not expected that any satisfactory necessary and sufficient conditions will be found for their existence. Hence the study of sufficient conditions of practical value is lively pursued.

A list of open problems in graph theory can be found in Bondy and Murty (2007).

Recommended Reading

Bang-Jensen J, Gutin G (2000) Digraphs: theory, algorithms and applications. Springer monographs in mathematics. Springer, London. http://www.imada.sdu.dk/Research/Digraphs/

Berge C (1976) Graphs and hypergraphs. North-Holland mathematical library, vol 6. North-Holland Publishing Company, Amsterdam

Bishop CM (2006) Pattern recognition and machine learning. Springer, New York

Bondy JA, Murty USR (2007) Graph theory. Springer

Chudnovsky M, Robertson N, Seymour P, Thomas R (2006) The strong perfect graph theorem. Ann Math 164:51–229

Diestel R (2005) Graph theory, 3rd edn. Springer. http://www.math.uni-hamburg.de/home/diestel/books/graph.theory/GraphTheoryIII.pdf

Emden-Weinert T. Graphs: theory–algorithms–complexity. http://people.freenet.de/Emden-Weinert/graphs.html

Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, New York

Garey MR, Johnson DS, Stockmeyer LJ (1976) Some simplified NP-complete graph problems. Theor Comput Sci 1:237–267

Gimbel J, Kennedy JW, Quintas LV (eds) (1993) Quo vadis, graph theory? North-Holland, Amsterdam/New York

Harary F (1969) Graph theory. Addison-Wesley, Reading

Jensen TR, Toft B (1995) Graph coloring problems. Wiley, New York

Locke SC. Graph theory. http://www.math.fau.edu/locke/graphthe.htm

Lovász L, Plummer MD (1986) Matching theory. Annals of discrete mathematics, vol 29. North Holland, Amsterdam/New York

Mohar B, Thomassen C (2001) Graphs on surfaces. John Hopkins University Press, Baltimore

Robertson N, Seymour PD (2004) Graph minors. XX. Wagner's conjecture. J Comb Theory Ser B 92(2): 325–357

Weisstein EW. Books about graph theory. http://www.ericweisstein.com/encyclopedias/books/GraphTheory.html

Greedy Search

Claude Sammut
The University of New South Wales, Sydney,
NSW, Australia

At each step in its search, a greedy algorithm makes the best decision it can at the time and continues without backtracking. For example, an algorithm may perform a ▸ general-to-specific search and at each step, commits itself to the specialization that best fits that training data, so far. It continues without backtracking to change any of its decisions. Greedy algorithms are used in many machine-learning algorithms, including decision tree learning (Breiman et al. 1984; Quinlan 1993) and ▸ rule learning algorithms, such as sequential covering.

Cross-References

▸ Rule Learning
▸ Learning as Search

Recommended Reading

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. Wadsworth International Group, Belmont

Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, San Mateo

Greedy Search Approach of Graph Mining

Lawrence Holder
Washington State University, Pullman, WA,
USA

Definition

▶ Greedy search is an efficient and effective strategy for searching an intractably large space when sufficiently informed heuristics are available to guide the search. The space of all subgraphs of a graph is such a space. Therefore, the greedy search approach of ▶ graph mining uses heuristics to focus the search toward subgraphs of interest while avoiding search in less interesting portions of the space. One such heuristic is based on the compression afforded by a subgraph; that is, how much is the graph compressed if each instance of the subgraph is replaced by a single vertex. Not only does compression focus the search, but it has also been found to prefer subgraphs of interest in a variety of domains.

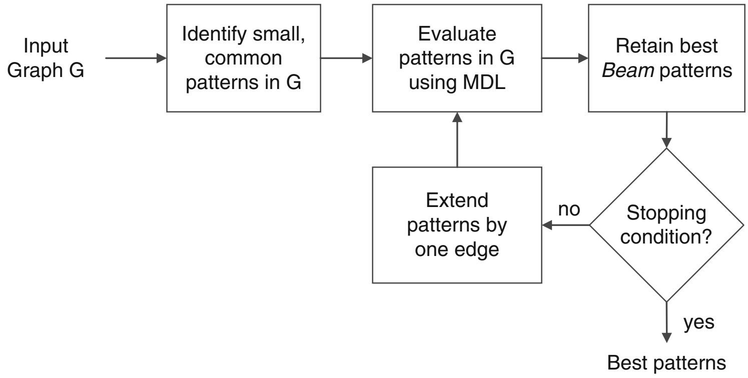
Motivation and Background

Many data mining and machine learning methods focus on the attributes of entities in the domain, but the relationships between these entities also represents a significant source of information, and ultimately, knowledge. Mining this relational information is an important challenge both in terms of representing the information and facing the additional computational obstacles of analyzing both entity attributes and relations. One efficient way to represent relational information is as a graph, where vertices in the graph represent entities in the domain, and edges in the graph represent attributes and relations among the entities. Thus, mining graphs is an important approach to extracting relational information. The main alternative to a graph-based representation is first-order logic, and the methods for mining this representation fall under the area of inductive logic programming. Here, the focus is on the graph representation.

Several methods have been developed for mining graphs (Washio and Motoda 2003), but most of these methods focus on finding the most frequent subgraphs in a set of graph transactions (e.g., FSG (Kuramochi and Karypis 2001), gSpan (Yan and Han 2002), Gaston (Nijssen and Kok 2004)) and use efficient exhaustive, rather than heuristic search. However, there are other properties besides frequency of a subgraph pattern that are relevant to many domains. One such property is the amount of compression afforded by the subgraph pattern, when each instance of the pattern is replaced by a single vertex. Searching for the most frequent subgraphs can be made efficient mainly through the exploitation of the downward closure property, which essentially says one can prune any extension of a subgraph that does not meet the minimum support frequency threshold. Unfortunately, the compression of a subgraph does not satisfy the downward closure property; namely, while a small extension of a subgraph may have less compression, a larger extension may have greater compression. Therefore, one cannot easily prune extensions and must search a larger portion of the space of subgraphs. Thus, one must resort to a greedy search method to search this space efficiently.

As with any greedy search approach, the resulting solution may sometimes be suboptimal, that is, the resulting subgraph pattern is not the pattern with maximum compression. The extent to which optimal solutions are missed depends on the type of greedy search and the strength of the heuristics used to guide the search. One approach is embodied in the graph-based induction (GBI) method (Matsuda et al. 2002; Yoshida et al. 1994). GBI continually compresses the input graph by identifying frequent triples of vertices, some of which may represent previously compressed portions of the input graph. Candidate triples are evaluated using a measure similar to information gain.

A similar approach recommended here is the use of a beam search strategy coupled with a compression heuristic based on the ▶ minimum description length (MDL) principle (Rissanen


Input Graph G
Identify small, common patterns in G
Evaluate patterns in G using MDL
Retain best *Beam* patterns
Stopping condition?
no
Extend patterns by one edge
yes
Best patterns

Greedy Search Approach of Graph Mining, Fig. 1 Structure of the greedy search approach of graph mining

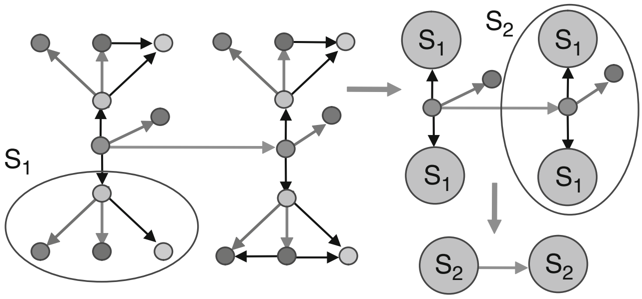
1989). The goal is to perform unsupervised discovery of a subgraph pattern that maximizes compression, which is essentially a tradeoff between frequency and size. Once the capability to find such a pattern exists, it can be used in an iterative discovery-and-compress fashion to perform hierarchical conceptual clustering, and it can be used to perform supervised learning, that is, find patterns that compress the positive graphs, but not the negative graphs. This approach has been well studied (Cook and Holder 2000, 2007; Gonzalez et al. 2002; Holder and Cook 2003; Jonyer et al. 2001; Kukluk et al. 2007) and has proven successful in several domains (Cook et al. 2001; Eberle and Holder 2006; Holder et al. 2005; You et al. 2006).

Structure of Learning System

Figure 1 depicts the structure of the greedy search approach of graph mining. The input data is a labeled, directed graph G. The search begins by identifying the set of small common patterns in G, that is, all vertices with unique labels having a frequency greater than one. The algorithm then iterates by evaluating the patterns according to the search heuristic, retaining the best patterns, and extending the best patterns by one edge until the stopping condition is met.

The search is guided by the minimum description length (MDL) principle, which seeks to minimize the description length of the entire data set. The evaluation heuristic based on the MDL principle assumes that the best pattern is the one that minimizes the description length of the input graph when compressed by the pattern. The description length of the pattern S given the input graph G is calculated as $DL(G, S) = DL(S) + DL(G|S)$, where $DL(S)$ is the description length of the pattern, and $DL(G|S)$ is the description length of the input graph compressed by the pattern. The search seeks a pattern S that minimizes $DL(G,S)$.

While several greedy search strategies apply here (e.g., hill climbing, stochastic), the strategy that has been found to work best is the ▸ beam search. Of the patterns currently under consideration, the system retains only the best *Beam* patterns, where *Beam* is a user-defined parameter. These patterns are then extended by one edge in all possible ways according to the input graph, the extended patterns are evaluated, and then again, all but the best *Beam* patterns are discarded. This process continues until the stopping condition is met. Several stopping conditions are applicable here, including a user-defined limit on the number of patterns considered, the exhaustion of the search space, or the case in which all extensions of a pattern evaluate to a lesser value than their parent pattern. Once meeting the stopping condition, the system returns the best patterns. Note that while the naïve approach to implementing this algorithm would require an NP-complete subgraph isomorphism procedure to collect the instances of each pattern, a more efficient approach takes advantage of the fact that new patterns are always one-edge extensions of existing patterns, and, therefore, the instances of the extended patterns can be identified by search-



Greedy Search Approach of Graph Mining, Fig. 2 Example of the greedy search approach of graph mining

ing the extensions of the parent's instances. This process does require several isomorphism tests, which is the computational bottleneck of the approach, but avoids the subgraph isomorphism problem.

Once the search terminates, the input graph can be compressed using the best pattern. The compression procedure replaces all instances of the pattern in the input graph by single vertices, which represent the pattern's instances. Incoming and outgoing edges to and from the replaced instances will point to, or originate from the new vertex that represents the instance. The algorithm can then be invoked again on this compressed graph.

Figure 2 illustrates the process on a simple example. The system discovers pattern S$_1$, which is used to compress the data. A second iteration on the compressed graph discovers pattern S$_2$. Because instances of a pattern can appear in slightly different forms throughout the data, an inexact graph match, based on graph edit distance, can be used to address noise by identifying similar pattern instances.

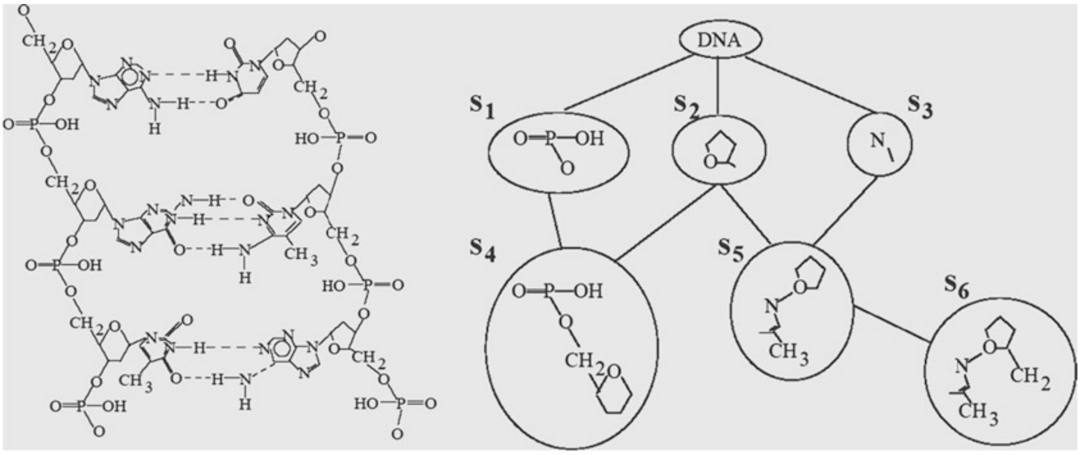
Graph-Based Hierarchical Conceptual Clustering

Given the ability to find a prevalent subgraph pattern in a larger graph and then compress the graph with this pattern, iterating over this process until the graph can no longer be compressed will produce a hierarchical, conceptual clustering of the input data (Jonyer et al. 2001). On the ith iteration, the best subgraph S_i is used to compress the input graph, introducing new vertices labeled S_i in the graph input to the next iteration. Therefore, any subsequently discovered subgraph S_j can be defined in terms of one or more of S_is, where $i < j$. The result is a *lattice*, where each cluster can be defined in terms of more than one parent subgraph. For example, Fig. 3 shows such a clustering done on a DNA molecule.

Graph-Based Supervised Learning

Extending a graph-based data mining approach to perform ▸ supervised learning involves the need to handle negative examples (focusing on the two-class scenario). In the case of a graph the negative information can come in three forms. First, the data may be in the form of numerous smaller graphs, or graph transactions, each labeled either positive or negative. Second, data may be composed of two large graphs: one positive and one negative. Third, the data may be one large graph in which the positive and negative labeling occurs throughout. The first scenario is closest to the standard supervised learning problem in that one has a set of clearly defined examples (Gonzalez et al. 2002). Let G^+ represent the set of positive graphs, and G^- represent the set of negative graphs. Then, one approach to supervised learning is to find a subgraph that appears often in the positive graphs, but not in the negative graphs. This amounts to replacing the information-theoretic measure with simply an error-based measure. This approach will lead the search toward a small subgraph that discriminates well. However, such a subgraph does not necessarily compress well, nor represent a characteristic description of the target concept.

One can bias the search toward a more characteristic description by using the information-theoretic measure to look for a subgraph that compresses the positive examples, but not the



Greedy Search Approach of Graph Mining, Fig. 3 Iterative application of the greedy search approach of graph mining yields the hierarchical, conceptual clustering on the right given an input graph representing the portion of DNA structure depicted on the left

negative examples. If $I(G)$ represents the description length (in bits) of the graph G, and $I(G|S)$ represents the description length of graph G compressed by subgraph S, then one can look for an S that minimizes $I(G^+|S) + I(S) + I(G^-) - I(G^-|S)$, where the last two terms represent the portion of the negative graph incorrectly compressed by the subgraph. This approach will lead the search toward a larger subgraph that characterizes the positive examples, but not the negative examples.

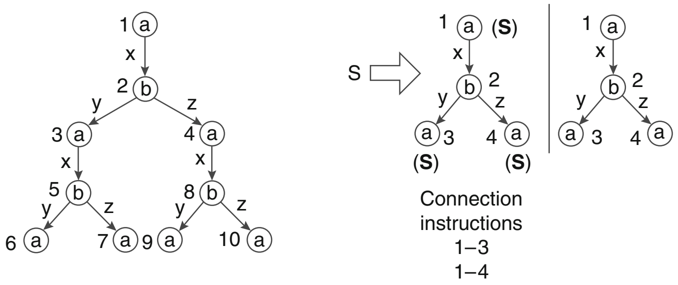
Finally, this process can be iterated in a set-covering approach to learn a disjunctive hypothesis. If using the error measure, then any positive example containing the learned subgraph would be removed from subsequent iterations. If using the information-theoretic measure, then instances of the learned subgraph in both the positive and negative examples (even multiple instances per example) are compressed to a single vertex. Note that the compression is a lossy one, that is, one does not keep enough information in the compressed graph to know how the instance was connected to the rest of the graph. This approach is consistent with the goal of learning general patterns, rather than mere compression.

Graph Grammar Inference

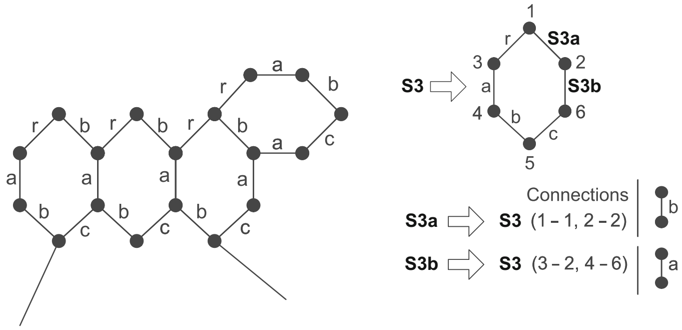
In the above algorithms the patterns are limited to non-recursive structures. In order to learn subgraph motifs, or patterns that can be used as the building blocks to generate arbitrarily large graphs, one needs the ability to learn graph grammars. The key to the inference of a graph grammar is the identification of overlapping structure. One can detect the possibility of a recursive graph-grammar production by checking if the instances of a pattern overlap. If a set of instances overlap by a single vertex, then one can propose a recursive node-replacement graph grammar production. Figure 4 shows an example of a node-replacement graph grammar (right) learned from a simple, repetitive input graph (left). The input graph in Fig. 4 is composed of three overlapping substructures. Based on how the instances overlap, one can also infer connection instructions that describe how the pattern can connect to itself. For example, the connection instructions in Fig. 4 indicate that the graph can grow by connecting vertex 1 of one pattern instance to either vertex 3 or vertex 4 of another pattern instance.

If a set of pattern instances overlap by an edge, then one can propose a recursive edge-replacement graph grammar production. Figure 5 shows an example of an edge-replacement graph grammar (right) learned from the input graph (left). Connection instructions describe how the motifs can connect via the edge labeled “a” or the edge labeled “b.”

Apart from the inclusion of recursive patterns, the greedy search approach of graph mining is



Greedy Search Approach of Graph Mining, Fig. 4 The node-replacement graph grammar (*right*) inferred from the input graph (*left*). The connection instructions indicate how the pattern can connect to itself



Greedy Search Approach of Graph Mining, Fig. 5 The edge-replacement graph grammar (*right*) inferred from the input graph (*left*). The connection instructions indicate how the pattern can connect to itself

unchanged. Both recursive and non-recursive patterns are evaluated according to their ability to compress the input graph using the MDL heuristic. After several iterations of the approach, the result is a graph grammar consisting of recursive and non-recursive productions that both describe the input graph and provide a mechanism for generating graphs with similar properties.

Programs and Data

Most of the aforementioned functionality has been implemented in the SUBDUE graph-based pattern learning system. The SUBDUE source code and numerous sample graph data files are available at http://www.subdue.org.

Applications

Many relational domains, from chemical molecules to social networks, are naturally represented as a graph, and a graph mining approach is a natural choice for extracting knowledge from such data. Three such applications are described below.

A huge amount of biological data that has been generated by long-term research encourages one to move one's focus to a systems-level understanding of bio-systems. A biological network, containing various biomolecules and their relationships, is a fundamental way to describe bio-systems. Multi-relational data mining finds the relational patterns in both the entity attributes and relations in the data. A graph consisting of vertices and edges between these vertices is a natural data structure to represent biological networks. The greedy search approach of graph mining has been applied to find patterns in metabolic pathways (You et al. 2006). Graph-based supervised learning finds the unique substructures in a specific type of pathway, which help one understand better how pathways differ. Unsupervised learning shows hierarchical clusters that describe the common substructures in a specific type of pathway, which allow one to better understand the common features in pathways.

Social network analysis is the mapping and measuring of relationships and flows between people, organizations, computers, or other information processing entities. Such analysis is naturally done using a graphical representation of the domain. The greedy approach of graph mining has been applied to distinguish between criminal and legitimate groups based on their mode of communication (Holder et al. 2005). For example, terrorist groups tend to exhibit communications chains; whereas, legitimate groups (e.g., families) tend to exhibit more hub-and-spoke communications.

▶ Anomaly detection is an important problem for detecting fraud or unlawful intrusions. However, anomalies are typically rare and, therefore, present a challenge to most mining algorithms that rely on regularity and frequency to detect patterns. With the graph mining approach's ability to iteratively compress away regularity in the graph, what is left can be construed as anomalous. To distinguish this residual structure from noise, one can compare its regularity with the probability that such structure would appear randomly. The presence of rare structure that is unlikely to appear by chance suggests an anomaly of interest. Furthermore, most fraudulent activity attempts to disguise itself by mimicking legitimate activity. Therefore, another method for finding such anomalies in graphs is to first find the normative pattern using the greedy search approach of graph mining and then find unexpected deviations to this normative pattern. This approach has been applied to detect anomalies in cargo data (Eberle and Holder 2006).

Future Directions

One of the main challenges in approaches to graph mining is scalability. Since most relevant graph operations (e.g., graph and subgraph isomorphism) are computationally expensive, they can be applied to only modest-sized graphs that can fit in the main memory. Clearly, there will always be graphs larger than can fit in main memory, so efficient techniques for mining in such graphs are needed. One approach is to keep the graph in a database and translate graph mining operations into database queries. Another approach is to create abstraction hierarchies of large graphs so that mining can occur at higher-level, smaller graphs to identify interesting regions of the graph before descending down into more specific graphs. Traditional high-performance computing techniques of partitioning a problem into subproblems, solving the subproblems, and then recomposing a solution do not always work for graph mining problems, because partitioning the problem means breaking links which may later turn out to be important. New techniques and architectures are needed to improve the scalability of graph mining operations.

Another challenge for graph mining techniques is dynamic graphs. Most graphs represent data that can change over time. For example, a social network can change as people enter and leave the network, new links are established and old links are discarded. First, one would like to be able to mine for static patterns in the presence of the changing data, which will require incremental approaches to graph mining. Second, one would like to mine patterns that describe the evolution of the graph over time, which requires mining of time slice graphs or the stream of graph transaction events. Third, the dynamics can reside in the attributes of entities (e.g., changing concentrations of an enzyme in a metabolic pathway), in the relation structure between entities (e.g., new relationships in a social network), or both. Research is needed on efficient and effective techniques for mining dynamic graphs.

Cross-References

▶ Grammatical Inferences

Recommended Reading

Cook D, Holder L (2000) Graph-based data mining. IEEE Intell Syst 15(2):32–41

Cook D, Holder L (eds) (2007) Mining graph data. Wiley, New Jersey

Cook D, Holder L, Su S, Maglothin R, Jonyer I (2001) Structural mining of molecular biology data. IEEE Eng Med Biol Spec Issue Genomics Bioinform 20(4):67–74

Eberle W, Holder L (2006) Detecting anomalies in cargo shipments using graph properties. In: Proceedings of the IEEE intelligence and security informatics conference, San Diego, May 2006

Gonzalez J, Holder L, Cook D (2002) Graph-based relational concept learning. In: Proceedings of the nineteenth international conference on machine learning, Sydney, July 2002

Holder L, Cook D (2003) Graph-based relational learning: current and future directions. ACM SIGKDD Explor 5(1):90–93

Holder L, Cook D, Coble J, Mukherjee M (2005) Graph-based relational learning with application to security. Fundamenta Informaticae, Spec Issue Min Graphs Trees Seq 66(1–2):83–101

Jonyer I, Cook D, Holder L (2001) Graph-based hierarchical conceptual clustering. J Mach Learn Res 2:19–43

Kukluk J, Holder L, Cook D (2007) Inference of node replacement graph grammars. Intell Data Anal 11(4):377–400

Kuramochi M, Karypis G (2001) Frequent subgraph discovery. In: Proceedings of the IEEE international conference on data mining (ICDM), San Jose, pp 313–320

Matsuda T, Motoda H, Yoshida T, Washio T (2002) Mining patterns from structured data by beam-wise graph-based induction. In: Proceedings of the fifth international conference on discovery science, Lubeck, pp 323–338

Nijssen S, Kok JN (2004) A quickstart in frequent structure mining can make a difference. In: Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining (KDD), Seattle, (pp 647–652)

Rissanen J (1989) Stochastic complexity in statistical inquiry. World Scientific, New Jersey

Washio T, Motoda H (2003) State of the art of graph-based data mining. ACM SIGKDD Explor 5(1):59–68

Yan X, Han J (2002) gSpan: graph-based substructure pattern mining. In: Proceedings of the IEEE international conference on data mining (ICDM), Maebashi City, pp 721–724

Yoshida K, Motoda H, Indurkhya N (1994) Graph-based induction as a unified learning framework. J Appl Intell 4:297–328

You C, Holder L, Cook D (2006) Application of graph-based data mining to metabolic pathways. In: Workshop on data mining in bioinformatics, IEEE international conference on data mining, Hong Kong, Dec 2006

Group Detection

Hossam Sharara and Lise Getoor
University of Maryland, College Park, MD, USA

Synonyms

Community detection; Graph clustering; Modularity detection

Definition

Group detection can defined as the clustering of nodes in a graph into groups or communities. This may be a hard partitioning of the nodes, or may allow for overlapping group memberships. A community can be defined as a group of nodes that share dense connections among each other, while being less tightly connected to nodes in different communities in the network. The importance of communities lies in the fact that they can often be closely related to modular units in the system that have a common function, e.g., groups of individuals interacting with each other in a society (Girvan and Newman 2002), WWW pages related to similar topics (Flake et al. 2002), or proteins having the same biological function within the cell (Chen and Yuan 2006).

Motivation and Background

The work done in group detection goes back as early as the 1920s when Stuart Rice clustered data by hand to investigate political blocks (Rice 1927). Another early example is the work of George Homans (1950) who illustrated how simple rearrangement of the rows and columns of data matrices helped to reveal their underlying structure. Since then, group detection has attracted researchers from different areas such as sociology, mathematics, physics, marketing, statistics, and computer science.

Group detection techniques vary from simple similarity-based ▸ clustering algorithms that follow the classical assumption that the data points are independent and identically distributed, to more advanced techniques that take into consideration the existing relationships between nodes in addition to their attributes, and try to characterize the different distributions present in the data.

Theory Solution

A network is defined as a graph $G = (V, E)$ consisting of a set of nodes $v \in V$, and a set of edges $e \in E$. In the case of weighted networks,

$w(v_i, v_j)$ denotes the weight of the edge connection nodes v_i and v_j. A community, or a group, C is a subgraph $C(V', E')$ of the original graph $G(V, E)$ whose nodes and edges are subsets of the original graph's nodes and edges; i.e., $V' \subset V$ and $E' \subset E$.

Following the definition of the community, we can expect that all the vertices in any community must be connected by a path within the same community. This property is referred to in literature as *connectedness*, which implies that in the case of disconnected graphs, we can analyze each connected component separately, as communities cannot span different components.

Another important property that follows from the definition of a community is that the group of vertices within a community should share denser connections among each other, and fewer connections with the other vertices in the network. To quantify this measure, the link density of a group $\delta(C)$ is defined as the ratio between the number of internal edges in that group and the maximum number of possible internal edges:

$$\delta(C) = \frac{|E'|}{|V'| \times (|V'| - 1)/2} \qquad (1)$$

Thus, for any community C, we require that $\delta(C) > \delta(G)$; where $\delta(G)$ is the average link density of the whole network. Similarly, the average link density between different communities, calculated using the ratio between the number of edges emanating from a group and terminating in another, and the maximum number possible of such edges, should generally be low.

Approaches

Beyond the intuitive discussion above, the precise definition of what constitutes a community involves multiple aspects. One important aspect is whether communities form hard partitions of the graph or nodes can belong to several communities. Overlapping communities do commonly occur in natural settings, especially in social networks. Currently, only a few methods are able to handle overlapping communities (Palla et al. 2005).

Other aspects should also be taken into consideration when defining community structure, such as whether link weights and/or directionalities are utilized, and whether the definition allows for hierarchical community structure, which means that communities may be parts of larger ones. However, one of the most important aspect that comes into consideration in community detection is whether the definition depends on global or local network properties. The main difference between the two approaches is whether the communities are defined in the scope of the whole network structure, such as methods based on centrality measures (Girvan and Newman 2002), global optimization methods (Newman and Girvan 2004), spectral methods (Arenas et al. 2006), or information-theoretic methods (Rosvall and Bergstrom 2008). Local methods, on the other hand, define communities based on purely local network structure, such as detecting cliques of different sizes, clique percolation method (Palla et al. 2005), and subgraph fitness method (Lancichinetti et al. 2009).

Local Techniques

Local methods for community detection basically rely on defining a set of properties that should exist in a community, then finding maximal subgraphs for which these set of properties hold. This formulation corresponds to finding maximal cliques in the network, where a clique is a subgraph in which all its vertices are directly connected.

However, there are some issues that rises from the previous formulation. First, finding cliques in a graph is an NP-Complete problem, thus most solutions will be approximate based on heuristic methods. Another more semantic issue is the interpretation of communities, especially in the context of social networks, where different individuals have different centralities within their corresponding groups, contradicting with the degree symmetry of the nodes in cliques. To overcome these drawbacks, the notion of a clique is relaxed to n-clique, which is a maximal subgraph where each pair of vertices are at most n-steps apart from each other.

Clustering Techniques

► Data clustering is considered one of the earliest techniques for revealing group structure, where data points are grouped based on the similarity between their corresponding features according to a given similarity measure. The main objective of traditional clustering methods is to obtain clusters or groups of data points possessing high intra-cluster similarity and low inter-cluster similarity. Classical data clustering techniques can be divided into partition-based methods such as k-means clustering (MacQueen 1967), spectral clustering algorithms (Alpert et al. 1999), and hierarchical clustering methods (Hartigan 1975), which are the most popular and the most commonly used in many fields.

One of the main advantages of the hierarchical clustering techniques is their ability to provide multiple resolutions at which the data can be grouped. In general, hierarchical clustering can be divided into agglomerative and divisive algorithms. The agglomerative algorithm is a greedy bottom-up one that starts with clusters including single data points then successively merge the pairs of clusters with the highest similarity. Divisive algorithms work in an opposite direction, where initially all the data points are regarded as one cluster, which is successively divided into smaller ones by splitting groups of nodes having the lowest similarity. In both algorithms, clusters are represented as a dendrogram, whose depths indicate the steps at which two clusters are joined. This representation clarifies which communities are built up from smaller modules, and how these smaller communities are organized, which can be particularly useful in the case of the presence of a normal hierarchy of community structure in the data. Hierarchical clustering techniques can easily be used in network domains, where data points are replaced by individual nodes in the network, and the similarity is based on edges between them.

Centrality-Based Techniques

One of the methods for community detection that is based on the global network structure is the one proposed by Girvan and Newman (2002), where they proposed an algorithm based on the betweenness centrality of edges to be able to recover the group structure within the network. Betweenness centrality is a measure of centrality of nodes in networks, defined for each node as the number of shortest paths between pairs of nodes in the network that run through it. The Girvan–Newman algorithm extended this definition for edges in the network as well, where the betweenness centrality of an edge is defined as the number of shortest paths between pairs of nodes that run along it.

The basic idea behind the algorithm is exploiting the fact that the number of edges connecting nodes from different communities is sparse. Following from that, all shortest paths between nodes from different communities should pass along one of these edges, increasing their edge betweenness centrality measure. Therefore, by following a greedy approach and removing edges with highest betweenness centrality from the network successively, the underlying community structure will be revealed. One of the major drawbacks of the algorithm is the time complexity, which is $O(|E|^2|V|)$ generally, and $O(|V|^3)$ for sparse networks. The fact that the edge betweenness needs only to be recalculated only for the edges affected by the edge removal can be factored in, which makes the algorithm efficient in sparse networks with strong community structure, but not very efficient on dense networks.

Modularity-Based Techniques

The concept of modularity was introduced by Newman and Girvan (2004) as a measure to evaluate the quality of a set of extracted communities in a network, and has become one of the most popular quality functions used for community detection. The basic idea is utilizing a *null model*: a network having the same set of nodes as the original one, but with random edges placed between them taking into account preserving the original node degrees. The basic idea is that the created random network is expected to contain no community structure, thus by comparing the number of edges within the extracted communities against the expected number of edges in the same communities from the random network, we can judge the quality of the extracted community structure. More specifically, the modularity Q is defined as follows

$$Q = \frac{1}{2|E|} \sum_{ij} \left[A_{ij} - \frac{\deg(i) \times \deg(j)}{2|E|} \right] \delta_k(c_i, c_j) \quad (2)$$

where A_{ij} is the element of the adjacency matrix of the network denoting the number of edges between nodes i and j, $\deg(i)$ and $\deg(j)$ are the degrees of nodes i and j respectively, c_i and c_j are the communities to which nodes i and j belong respectively, and δ_k refers to the kronecker delta. The summation runs over all pairs of nodes within the same community.

Clearly, a higher modularity value indicates that the average link density within the extracted community is larger than that of the random network where no community structure is present. Thus, modularity maximization can be used as the objective for producing high-quality community structure. However, modularity maximization is an NP-hard problem. Nevertheless, there have been several algorithms for finding fairly good approximations of the modularity maximum in reasonable amount of time.

One of the first modularity maximization algorithms was introduced by Newman in 2004. It is a greedy hierarchical agglomerative clustering algorithm, which starts with individual nodes and merges them in the order of increasing the overall modularity of the resulting configuration. The time complexity of this greedy algorithm is $O(|V|(|E| + |V|))$ or $O(|V|^2)$ for sparse networks, which enables the user to run community detection on large networks in a reasonable amount of time.

Issues

One of the main issues with the methods of group detection in network setting is the focus on the network structure, without taking into consideration other properties of nodes and edges in the network. This issue often results in a lack of correspondence between the extracted communities and the functional groups in the network (Shalizi et al. 2007). This also leads to another common problem which is how to validate the resulting communities produced by any of the proposed techniques.

Although in network settings there are often different types of interactions between entities of different natures, most group detection methods work on single-mode networks, which have just a single node and edge type. Fewer works focus on finding groups in more complex, multimodal settings, where nodes from different types have multiple types of interactions with each other. One of the most common approaches to deal with these types of networks is projecting them into a series of individual graphs for each node type. However, this approach results in losing some of the information that could have been retained by operating collectively on the original multi-relational network.

Another issue also gaining interest is developing methods for group detection in dynamic network settings (Tantipathananandh and Berger-Wolf 2009), where the underlying network structure changes over time. Most of the previous work on group detection focused on static networks, and handles the dynamic case by either analyzing a snapshot of the network at a single point in time, or aggregating all interactions over the whole time period. Both approaches do not capture the dynamics of change in the network structure, which can be an important factor in revealing the underlying communities.

Cross-References

- ▸ Graph Clustering
- ▸ Graph Mining

Recommended Reading

Alpert C, Kahng A, Yao S (1999) Spectral partitioning: the more eigenvectors, the better. Discret Appl Math 90:3–26

Arenas A, Daz-Guilera A, Prez-Vicente CJ (2006) Synchronization reveals topological scales in complex networks. Phys Rev Lett 96(11):114102

Chen J, Yuan B (2006) Detecting functional modules in the yeast protein–protein interaction network. Bioinformatics 22(18):2283–2290

Flake GW, Lawrence S, Giles CL, Coetzee F (2002) Self-organization and identification of web communities. IEEE Comput 35:66–71

Girvan M, Newman MEJ (2002) Community structure in social and biological networks. Proc Natl Acad Sci 99:7821–7826

Hartigan JA (1975) Clustering algorithms. Wiley, New York

Homans GC (1950) The human group. Harcourt, Brace, New York

Lancichinetti A, Fortunato S, Kertesz J (2009) Detecting the overlapping and hierarchical community structure in complex networks. N J Phys 11:033015

MacQueen JB (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of fifth Berkeley symposium on mathematical statistics and probability, vol 1. University of California Press, Berkeley, pp 281–297

Newman MEJ (2004) Fast algorithm for detecting community structure in networks. Phys Rev E 69(6):066133

Newman MEJ, Girvan M (2004) Finding and evaluating community structure in networks. Phys Rev E 69:026113

Palla G, Dernyi I, Farkas I, Vicsek T (2005) Uncovering the overlapping community structure of complex networks in nature and society. Nature 435(7043):814–818

Rice SA (1927) The identification of blocs in small political bodies. Am Pol Sci Rev 21: 619–627

Rosvall M, Bergstrom CT (2008) Maps of random walks on complex networks reveal community structure. Proc Natl Acad Sci 105: 1118–1123

Shalizi CR, Camperi MF, Klinkner KL (2007) Discovering functional communities in dynamical networks. In: Statistical network analysis: models, issues, and new directions. Springer, Berlin, pp 140–157

Tantipathananandh C, Berger-Wolf TY (2009) Algorithms for identifying dynamic communities. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining, Paris. ACM, New York

Grouping

▸ Categorical Data Clustering

Growing Set

Definition

A growing set is a subset of a ▸ training set containing data that are used by a learning system to develop models that are then evaluated against a ▸ pruning set.

Cross-References

▸ Data Set

Growth Function

▸ Shattering Coefficient

H

Hebb Rule

▶ Biological Learning: Synaptic Plasticity, Hebb Rule and Spike Timing Dependent Plasticity

Hebbian Learning

Synaptic weight changes depend on the joint activity of the ▶ presynaptic and postsynaptic neurons.

Cross-References

▶ Biological Learning: Synaptic Plasticity, Hebb Rule and Spike Timing Dependent Plasticity

Heuristic Rewards

▶ Reward Shaping

Hidden Markov Models

Antal van den Bosch
Centre for Language Studies, Radboud
University, Nijmegen, The Netherlands

Abstract

Starting from the concept of regular Markov models we introduce the concept of hidden Markov model, and the issue of estimating the output emission and transition probabilities between hidden states, for which the Baum-Welch algorithm is the standard choice. We mention typical application in which hidden Markov models play a central role, and mention a number of popular implementations.

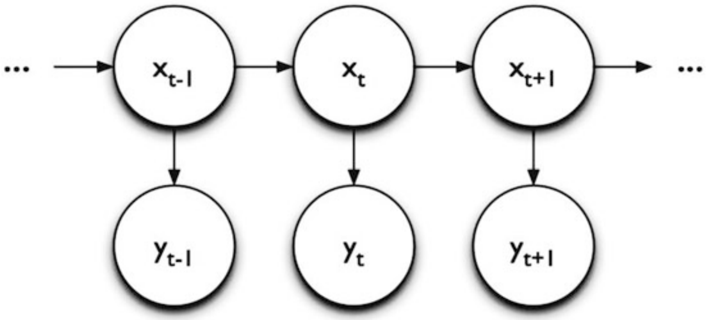
Definition

Hidden Markov models (HMMs) form a class of statistical models in which the system being modeled is assumed to be a Markov process with hidden states. From observed output sequences generated by the Markov process, both the output emission probabilities from the hidden states and the transition probabilities between the hidden states can be estimated with dynamic programming methods. The estimated model parameters can then be used for various sequence analysis purposes.

Motivation and Background

The states of a regular Markov model, named after Russian mathematician Andrey Markov (1865–1922), are directly observable; hence its only parameters are the state transition probabilities. In many real-world cases, however, the states of the system that one wants to model are not directly observable. For instance, in speech recognition, the audio is the observable stream, while the goal is to discover the phonemes (the

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1



Hidden Markov Models, Fig. 1 Architecture of a hidden Markov model

categorical elements of speech) that emitted the audio. Hidden Markov models offer one type of architecture to estimate hidden states through indirect means. Dynamic programming methods have been developed that can estimate both the output emission probabilities and the transition probabilities between the hidden states, either from observations of output sequences only (an unsupervised learning setting) or from pairs of aligned output sequences and gold-standard hidden sequences (a supervised learning setting).

Structure of the Learning System

Figure 1 displays the general architecture of a hidden Markov model. Each circle represents a variable x_i or y_i occurring at time i; x_t is the discrete value of the hidden variable at time t. The variable y_t is the output variable observed at the same time t, said to be emitted by x_t. Arrows denote conditional dependencies. Any hidden variable is only dependent on its immediate predecessor; thus, the value of x_t is only dependent on that of x_{t-1} occurring at time $t-1$. This deliberate simplicity is referred to as the Markov assumption. Analogously, observed variables such as y_t are conditionally dependent only on the hidden variables occurring at the same time t, i.e., x_t in this case.

Typically, a start state x_0 is used as the first hidden state (not conditioned by any previous state), as well as an end state x_{n+1} that closes the hidden state sequence of length n. Start and end states usually emit meta-symbols signifying the "start" and "end" of the sequence.

An important constraint on the data that can in principle be modeled in a hidden Markov model is that the hidden and output sequences need to be discrete, aligned (i.e., one y_t for each x_t), and of equal length. Sequence pairs that do not conform to these constraints need to be discretized (e.g., in equal-length time slices) or aligned where necessary.

Training and Using Hidden Markov Models

Hidden Markov models can be trained both in an unsupervised and a supervised fashion. First, when only observed output sequences are available for training, the model's conditional probabilities from this indirect evidence can be estimated through the Baum-Welch algorithm (Baum et al. 1970), a form of unsupervised learning, and an instantiation of the expectation-maximization (EM) algorithm (Dempster et al. 1977).

When instead aligned sequences of gold-standard hidden variables and output variables are given as supervised training data, both the output emission probabilities and the state transition probabilities can be straightforwardly estimated from frequencies of co-occurrence in the training data.

Once trained, it is possible to find the most likely sequence of hidden states that could have generated a particular (test) output sequence by the Viterbi algorithm (Viterbi 1967).

Applications of Hidden Markov Models

Hidden Markov models are known for their successful application in pattern recognition tasks such as speech recognition (Rabiner 1989) and DNA sequencing (Kulp et al. 1996) but also in sequential pattern analysis tasks such as in part-of-speech tagging (Church 1988).

Their introduction in speech recognition in the 1970s (Jelinek 1998) led the way toward the introduction of stochastic methods in general in the field of natural language processing in the 1980s and 1990s (Charniak 1993; Manning and Schütze 1999) and into text mining and information extraction in the late 1990s and onward (Freitag and McCallum 1999). In a similar way, hidden Markov models started to be used in DNA pattern recognition in the mid-1980s and have gained widespread usage throughout bioinformatics since (Durbin et al. 1998; Burge and Karlin 1997).

Programs

Many implementations of hidden Markov models exist. Three noteworthy packages are the following:

- UMDHMM by Tapas Kanungo. Implements the forward-backward, Viterbi, and Baum-Welch algorithms (Kanungo 1999)
- JAHMM by Jean-Marc François. A versatile Java implementation of algorithms related to hidden Markov models (François 2006)
- HMMER by Sean Eddy. An implementation of profile HMM software for protein sequence analysis (Eddy 2007)

Cross-References

▶ Baum-Welch Algorithm
▶ Bayesian Methods
▶ Expectation Maximization Clustering
▶ Markov Process
▶ Viterbi Algorithm

Recommended Reading

Baum LE, Petrie T, Soules G, Weiss N (1970) A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. Ann Math Stat 41(1):164–171

Burge C, Karlin S (1997) Prediction of complete gene structures in human genomic DNA. J Mol Biol 268:78–94

Charniak E (1993) Statistical language learning. The MIT Press, Cambridge, MA

Church KW (1988) A stochastic parts program and noun phrase parser for unrestricted text. In: Proceedings of the second conference on applied natural language processing, Austin, pp 136–143

Dempster A, Laird N, Rubin D (1977) Maximum likelihood from incomplete data via the EM algorithm. J R Stat Soc Ser B 39(1):1–38

Durbin R, Eddy S, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge

Eddy S (2007) HMMER. http://hmmer.org/

François J-M (2006) JAHMM. https://code.google.com/p/jahmm/

Freitag D, McCallum A (1999) Information extraction with HMM structures learned by stochastic optimization. In: Proceedings of the national conference on artificial intelligence. The MIT Press, Cambridge, MA, pp 584–589

Jelinek F (1998) Statistical methods for speech recognition. The MIT Press, Cambridge, MA

Kanungo T (1999) UMDHMM: hidden Markov model toolkit. In: Kornai A (ed) Extended finite state models of language. Cambridge University Press, Cambridge. http://www.kanungo.us/software/software.html

Kulp D, Haussler D, Reese MG, Eeckman FH (1996) A generalized hidden Markov model for the recognition of human genes in DNA. Proc Int Conf Intell Syst Mol Biol 4:134–142

Manning C, Schütze H (1999) Foundations of statistical natural language processing. The MIT Press, Cambridge, MA

Rabiner LR (1989) A tutorial on hidden Markov models and selected applications in speech recognition. Proc IEEE 77(2):257–286

Viterbi AJ (1967) Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Trans Inf Theory 13(2):260–269

Hierarchical Reinforcement Learning

Bernhard Hengst
University of New South Wales, Sydney, NSW, Australia

Definition

Hierarchical reinforcement learning (HRL) decomposes a ▶ reinforcement learning problem into a hierarchy of subproblems or subtasks such that higher-level parent-tasks invoke lower-level child tasks as if they were primitive actions. A decomposition may have multiple levels of hierarchy. Some or all of the subproblems can themselves be reinforcement learning problems. When a parent-task is formulated as a reinforcement learning problem it is commonly formalized as a semi-Markov decision problem because its actions are child-tasks that persist for an extended period of time. The advantage of hierarchical decomposition is a reduction in computational complexity if the overall problem can be represented more compactly and reusable subtasks learned or provided independently. While the solution to a HRL problem is optimal given the constraints of the hierarchy there are no guarantees in general that the decomposed solution is an optimal solution to the original reinforcement learning problem.

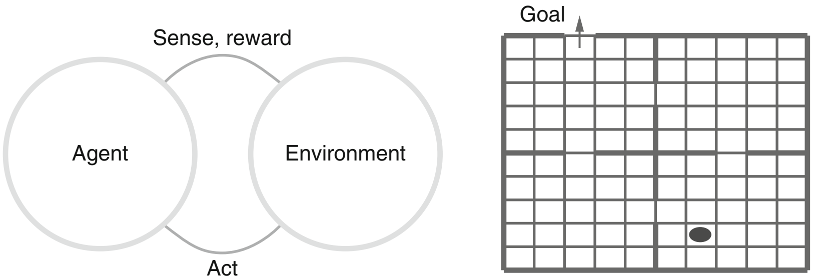
Motivation and Background

Bellman's "curse of dimensionality" beleaguers reinforcement learning because the problem representation grows exponentially in the number of state and action variables. The complexity we encounter in natural environments has a property, near decomposability, that may be exploited using hierarchical models to greatly simplify our understanding and control of behavior. Human societies have used hierarchical organizations to solve complex tasks dating back to at least Egyptian times. It seems natural, therefore, to introduce hierarchical structure into reinforcement learning to solve more complex problems.

When large problems can be decomposed hierarchically there may be improvements in the time and space complexity for both learning and execution of the overall task. Hierarchical decomposition is a divide-and-conquer strategy that solves the smaller subtasks and puts them back together for a more cost-effective solution to the larger problem. The subtasks defined over the larger problem are stochastic macro-operators that execute their policy until termination. If there are multiple ways to terminate a subtask the optimal subtask policy will depend on the context in which the subtask is invoked. Subtask policies usually persist for multiple time-steps and are hence referred to as *temporally extended actions*. Temporally extended actions have the potential to transition through a much smaller "higher-level" state-space, reducing the size of the original problem. For example, navigating through a house may only require room states to represent the abstract problem if room-leaving temporally extended actions are available to move through each room. A room state in this example is referred to as an *abstract state* as the detail of the exact position in the room is abstracted away. Hierarchical reinforcement learning can also provide opportunities for subtask reuse. If the rooms are similar, the policy to leave a room will only need to be learnt once and can be transferred and reused.

Early developments of hierarchical learning appeal to analogies of boss – subordinate models. Ashby (1956) discusses the "amplification" of regulation in very large systems through hierarchical control – a doctor looks after a set of mechanics who in turn maintain thousands of air-conditioning systems. Watkins (1989) used a navigator – helmsman hierarchical control example to illustrate how reinforcement learning limitations may be overcome. Early examples of hierarchical reinforcement learning include Singh's Hierarchical-DYNA (Dyna, a class of architectures for intelligent systems based on approximating dynamic programming methods. Dyna architectures integrate trial-and-error (reinforcement) learning and execution-time planning into a single process operating alternately on the world and on a learned model of the world Sutton et al. 1999) (Singh 1992), Kaelbling's Hierarchical Distance to Goal (HDG) (Kaelbling 1993), and Dayan and Hinton's Feudal reinforcement learning (Dayan and Hinton 1992). The latter explains hierarchical structure in terms of a management hierarchy. The example has four hierarchical levels and employs abstract states, which they refer to as "information hiding".

Close to the turn of the last century three approaches to hierarchical reinforcement learning were developed relatively independently: Hierarchies of Abstract Machines (HAMs) (Parr



Hierarchical Reinforcement Learning, Fig. 1 *Left*: The agent view of reinforcement learning. *Right*: A four-room environment with the agent in one of the rooms show as a *solid black* oval

and Russell 1997); the Options framework (Sutton et al. 1999); and MAXQ value function decomposition (Dietterich 2000). Each approach has different emphases, but a common factor is the use of temporally extended actions and the formalization of HRL in terms of semi-Markov decision process theory (Puterman 1994) to solve the higher-level abstract reinforcement learning problem.

Hierarchical reinforcement learning is still an active research area. More recent extensions include: continuous state-space; concurrent actions and multi-agency; use of average rewards (Ghavamzadeh and Mahadevan 2002); continuing problems; policy-gradient methods; partial-observability and hierarchical memory; factored state-spaces and graphical models; and basis functions. Hierarchical reinforcement learning also includes hybrid approaches such as Ryan's reinforcement learning teleo-operators (RL-TOPs) (Ryan and Reid 2000) that combines planning at the top level and reinforcement learning at the more stochastic lower levels. Please see Barto and Mahadevan (2003) for a survey of recent advances in hierarchical reinforcement learning. More details can be found in the section on recommended reading.

In most applications the structure of the hierarchy is provided as background knowledge by the designer. Some researchers have tried to learn the hierarchical structure from the agent–environment interaction. Most approaches look for subgoals or subtasks that try to partition the problem into near independent reusable subproblems.

Structure of Learning System

Structure of HRL

The agent view of reinforcement learning illustrated on the left in Fig. 1 shows an agent interacting with an environment. At regular time-steps the agent takes actions in the environment and receives sensor observations and rewards from the environment. A hierarchical reinforcement learning agent is given or discovers background knowledge that explicitly or implicitly provides a decomposition of the environment. The agent exploits this knowledge to solve the problem more efficiently by finding an action policy to optimize a measure of future reward, as for reinforcement learning.

We will motivate the machinery of hierarchical reinforcement learning with the simple example shown in Fig. 1 (right). This diagram shows a four-room house with doorways between adjoining rooms and a doorway in the top left room leading outside. Each cell represents a possible position of the agent. We assume the agent always starts in the bottom left room position as shown by the black oval. It is able to sense its position in the room and which room it occupies. It can move one step in any of the four compass directions each time-step. It also receives a reward of -1 at each time-step. The objective is to leave the house via the least-cost route. We assume that the actions are stochastic with an 80 % chance of moving in the intended direction and a 20 % chance of staying in place. Solving this problem in a straightforward manner using reinforcement learning requires storage for 400 Q values defined over 100 states and 4 actions.

If the state space is decomposed into the four identical rooms a hierarchical reinforcement learner could solve this problem more efficiently. For example, we could solve two subproblems. One that finds an optimal solution to leave a room to the North and another to leave a room to the West. When learning these subtasks, leaving a room in any other way is disallowed. Each of these subproblems requires storage for 100 Q values – 25 states and 4 actions.

We also formulate and solve a higher-level problem that consists of only the four rooms as states. These are abstract states because, as previously explained, the exact position in the room has been abstracted away. In each abstract state we allow a choice of only one or the other of the learnt room-leaving actions. These are temporally extended actions because, once invoked, they will usually persist for multiple time-steps until the agent exits the room. We proceed to solve this higher-level problem in the usual way using reinforcement learning. The proviso is that the reward on completing a temporally extended action is the sum of rewards accumulated since invocation of the subtask. The higher-level problem requires storage for only 8 Q values – 4 states and 2 actions.

Once learnt, execution of the higher-level policy will determine the optimal room-leaving action to invoke given the current room – in this case to leave the room via the West doorway. Control is passed to the room-leaving subtask that leads the agent out of the room through the chosen doorway. Upon leaving the room, the subtask is terminated and control is passed back to the higher level that chooses the next optimal room-leaving action until the agent finally leaves the house. The total number of Q values required for the hierarchical reinforcement formulation is 200 for the two subtasks and eight for the higher-level problem, a total of 208. This almost halves the storage requirements compared to the "flat" formulation with corresponding savings in time complexity. In this example, hierarchical reinforcement learning finds the same optimal policy that a less efficient reinforcement learner would find, but this is not always the case.

The above example hides many issues that hierarchical reinforcement learning needs to address, including: safe state abstraction; appropriately accounting for accumulated subtask reward when initial conditions change or rewards are discounted; optimality of the solution; and learning of the hierarchical structure itself. In the next sections we will touch on these issues as we discuss the semi-Markov decision problem formalism and review several approaches to hierarchical reinforcement learning.

Semi-Markov Decision Problem Formalism

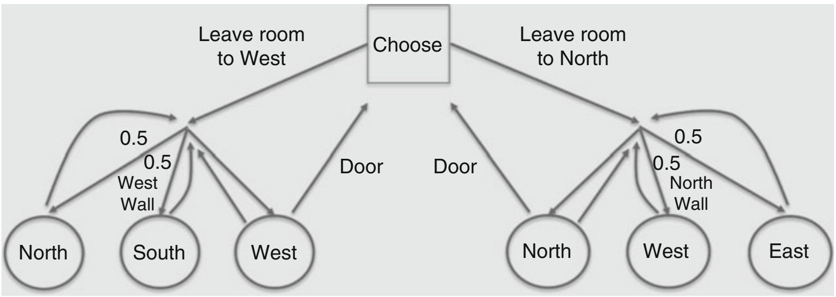
The common underlying formalism in hierarchical reinforcement learning is the semi-Markov decision process (SMDP). A SMDP generalizes a ▸ Markov decision process by allowing actions to be temporally extended. We will state the discrete time equations following Dietterich (2000), recognizing that in general SMDPs are formulated with real-time valued temporally extended actions (Puterman 1994).

Denoting the random variable N to be the number of time steps that a temporally extended action a takes to complete when it is executed starting in state s, the state transition probability function for the result state s' and the expected reward function are given by (1) and (2) respectively.

$$T_{ss'}^{N,a} = Pr\{s_{t+N} = s' | s_t = s, a_t = a\} \quad (1)$$

$$R_{ss'}^{N,a} = E\left\{ \sum_{n=1}^{N} \gamma^{n-1} r_{t+n} | s_t \right.$$
$$\left. = s, a_t = a, s_{t+N} = s' \right\} \quad (2)$$

$R_{ss'}^{N,a}$ is the expected sum of N future discounted rewards. The discount factor $\gamma \in [0, 1]$. When set to less than 1, γ insures that the value function will converge for continuing or infinite-horizon problems. The Bellman "backup" equations for the value function $V(s)$ for an arbitrary policy π and optimal policies (denoted by $*$) are similar to those for MDPs with the sum taken with respect to s' and N.



Hierarchical Reinforcement Learning, Fig. 2 An abstract machine for a HAM that provides routines for leaving rooms to the West and North of the house in Fig. 1 *right*

$$V^{\pi}_{m}(s) = \sum_{s',N} T^{N,\pi(s)}_{ss'} \left[R^{N,pi(s)}_{ss'} + \gamma^{N} V^{\pi}_{m}(s') \right] \tag{3}$$

$$V^{*}_{m}(s) = \max_{a} \sum_{s',N} T^{N,a}_{ss'} \left[R^{N,a}_{ss'} + \gamma^{N} V^{*}_{m}(s') \right] \tag{4}$$

For problems that are guaranteed to terminate, the discount factor γ can be set to 1. In this case the number of steps N can be marginalized out and the sum taken with respect to s alone. The above equations are then similar to the ones for MDPs with the expected primitive reward replaced with the expected sum of rewards to termination of the temporally extended action. All the methods developed for reinforcement learning using primitive actions work equally well for problems using temporally extended actions.

Approaches to Hierarchical Reinforcement Learning

Hierarchies of Abstract Machines (HAMs)

In the HAM approach to hierarchical reinforcement learning (Parr and Russell 1997), the designer specifies subtasks by providing stochastic finite state automata called abstract machines. While in practice several abstract machines may allow some to call others as subroutines (hence hierarchies of abstract machines), in principle this is equivalent to specifying one large abstract machine with two types of states. Action states, that specify the action to be taken given the state of the MDP to be solved and choice states with nondeterministic actions.

An abstract machine is a triple $\langle \mu, I, \delta \rangle$, where μ is a finite set of machine states, I is a stochastic function from states of the MDP to be solved to machine states that determines the initial machine state, and δ is a stochastic next-state function mapping machine states and MDP states to next machine states. The parallel action of the MDP and an abstract machine yields a discrete-time higher-level SMDP with the abstract machine's action states generating a sequence of temporally extended actions between choice states. Only a subset of states of the original MDP are associated with choice-points, potentially reducing the higher-level problem significantly.

Continuing with our four-room example, the abstract machine in Fig. 2 provides choices for leaving a room to the West or the North. In each room it will take actions that move the agent to a wall, and perform a random walk along the wall until it finds the doorway. Only five states of the original MDP are states of the higher-level SMDP. These states are the initial state of the agent and the states on the other side of doorways where the abstract machine enters choice states. Reinforcement learning methods update the value function for these five states in the usual way with rewards accumulated since the last choice state. The optimal policy consists of the three temporally extended actions sequentially leaving a room to the West, North, and North again.

Solving the SMDP will yield an optimal policy for the agent to leave the house subject to the constraints of the abstract machine. In this case it is not a globally optimal policy because a random walk along walls to find a doorway is inefficient.

The HAM approach is predicated on engineers and control theorists being able to design good controllers that will realize specific lower level behaviors. HAMs are a way to partially specify procedural knowledge to transform an MDP to a reduced SMDP. In the most general case HAMs can be Turing machines that execute any computable mapping of the agent's complete sensory-action history.

Options

For an MDP with finite states S and actions A, *options* generalize one-step primitive actions to include temporally extended actions (Sutton et al. 1999). Options consist of three components: a policy $\pi : S \times A \rightarrow [0, 1]$, a termination condition $\beta : S \rightarrow [0, 1]$, and an initiation set $I \subseteq S$. An option $\langle I, \pi, \beta \rangle$ is available in state s if and only if $s \in I$. If an option is invoked, actions are selected according to π until the option terminates according to β. These options are called Markov options because intra-option actions taken by policy π depend only on the current state s. It is possible to generalize options to semi-Markov options in which policies and termination conditions make their choices dependent on all prior events since the option was initiated. In this way it is possible, for example, to "time-out" options after some period of time has expired. For their most general interpretation, options and HAMs appear to have similar functionality, but different emphases.

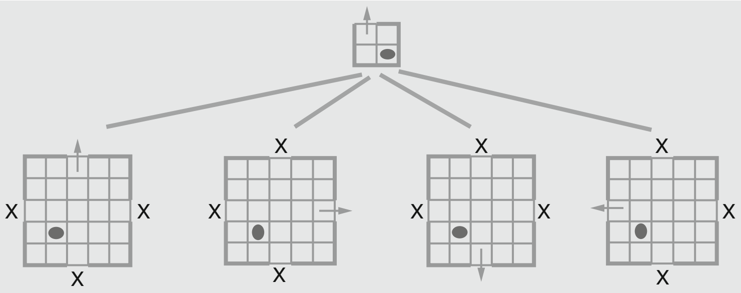
Options were intended to augment the primitive actions available to an MDP. The temporally extended actions executed by the options yield a SMDP. As for HAMs, if options replace primitive actions, the SMDP can be considerably reduced. There is debate as to benefits when primitive actions are retained. Reinforcement learning may be accelerated because the value function can be backed-up over greater distances in the state-space and the inclusion of primitive actions guarantees convergence to the globally optimal policy, but the introduction of additional actions increased the storage and exploration necessary.

In a similar four-room example to that of Fig. 1, the authors (Sutton et al. 1999) show how options can learn significantly faster proceeding on a room-by-room basis, rather than position by position. When the goal is not in a convenient location, able to be reached by the given options, it is possible to include primitive actions as special-case options and still accelerate learning for some problems. For example, with room-leaving options alone, it is not possible to reach a goal in the middle of a room. Primitive actions are required when the room containing the goal state is entered. Although the inclusion of primitive actions guarantees convergence to the globally optimal policy, this may create extra work for the learner.

MAXQ

The MAXQ (Dietterich 2000) approach to hierarchical reinforcement learning restricts subtasks to subsets of states, actions, and policy fragments of the original MDP without introducing extra state, as is possible with HAMs and semi-Markov options. The contribution of MAXQ is the decomposition of the value function over the hierarchy and provision of opportunities for state abstraction. An MDP is manually decomposed into a hierarchical directed acyclic graph of subtasks called a task-hierarchy. Each subtask is a smaller (semi-)MDP. In decomposing the MDP the designer specifies the active states and terminal states for each subtask. Terminal states are typically classed either as goal terminal states or non-goal terminal states. Using disincentives for non-goal terminal states, policies are learned for each subtask to encourage them to terminate in goal terminal states. The actions available in each subtask can be primitive actions or other (child) subtasks. Each sub-task can invoke any of its child subtasks as a temporally extended action. When a task enters a terminal state, it, and all its children, abort and return control to the calling subtask.

Figure 3 shows a task-hierarchy for the previous four-room problem. The four lower-level subtasks are sub-MDPs for a generic room, where a separate policy is learnt to exit a room by each of the four possible doorways. The arrow indicates a transition to a goal terminal state and the "×"s indicate non-goal terminal states. States, actions, transitions, and rewards are inherited



Hierarchical Reinforcement Learning, Fig. 3 A task-hierarchy decomposing the four-room problem in Fig. 1. The four lower-level subtasks are generic room-leaving sub-MDPs, one for leaving a room in each compass direction

from the original MDP. The rewards on transition to terminal states are engineered to encourage the agent to avoid non-goal terminal states and terminate in goal states. The higher-level problem (SMDP) consists of just four states representing the rooms. Any of the subtasks (room-leaving actions) can be invoked in any of the rooms.

A key feature of MAXQ is that it represents the value of a state as a decomposed sum of subtask completion values plus the value of the immediate primitive action. A completion value is the expected (discounted) cumulative reward to complete the subtask after taking the next (temporally extended) action when following a policy over subtasks. The sum includes all the tasks invoked on the path from the root task in the task hierarchy right down to the primitive action. For a rigorous mathematical treatment the reader is referred to Dietterich (2000). The Q function is expressed recursively (5) as the value for completing the subtask plus the completion value for the overall problem after the subtask has terminated. In this equation, i is the subtask identifier, s is the current state, action a is the child subtask (or primitive action), and π is a policy for each subtask.

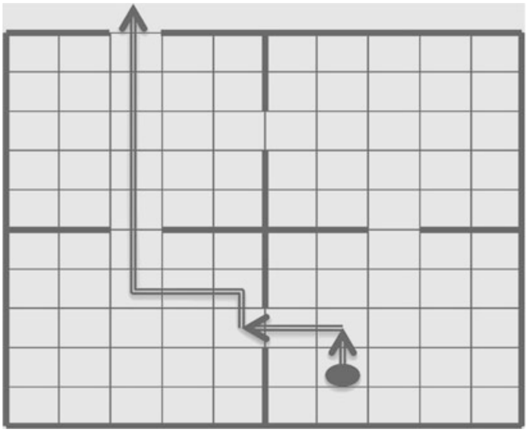
$$Q^{\pi}(i, s, a) = V^{\pi}(a, s) + C^{\pi}(i, s, a) \qquad (5)$$

We describe the basic idea for the task-hierarchy shown in Fig. 3 for the optimal policy. The value of the agent's state has three components determined by the two levels in the task-hierarchy plus a primitive action. For the agent state, shown in Fig. 4 by a solid black oval, the value function represents the expected reward for taking the next primitive action to the North, completing the lower-level subtask of leaving the room to the West, and completing the higher-level task of leaving the house. The benefit of decomposing the value function is that it can be represented much more compactly because only the completion values for non-primitive subtasks and primitive actions need be stored.

The example illustrates two types of state abstraction. As all the rooms are similar we can ignore the room identity when we learn intra-room navigation policies. Secondly, when future rewards are not discounted, the completion value after leaving a room is independent of the starting state in that room. These "funnel" actions allow the intra-room states to be abstracted into a single state for each room as far as the completion value is concerned. The effect is that the original problem can be decomposed into a small four-state SMDP at the top level and four smaller subtask MDPs.

Optimality

Hierarchical reinforcement learning can at best yield solutions that are hierarchically optimal, assuming convergence conditions are met, meaning that they are consistent with the task-hierarchy. MAXQ introduces another form of optimality – recursive optimality. MAXQ optimizes subtask policies to reach goal states ignoring the needs of their parent tasks. This has the advantage that subtasks can be reused in various contexts, but they may not therefore be optimal in each situation. A recursively optimal solution cannot be better than a hierarchical optimal solution. Both recursive and hierarchical optimality can be arbitrarily worse than the globally optimal solution if a designer chooses a poor HAM, option or hierarchical decomposition.



Hierarchical Reinforcement Learning, Fig. 4 The components of the decomposed value function for the agent following an optimal policy for the four-room problem in Fig. 1. The agent is shown as a *solid black* oval at the starting state

The stochastic nature of MDPs means that the condition under which a temporally abstract action is appropriate may have changed after the action's invocation and that another action may become a better choice because of "stochastic drift." A subtask policy proceeding to termination in this situation may be suboptimal. By constantly interrupting the subtask, as for example in HDG (Kaelbling 1993), a better subtask may be chosen. Dietterich calls this "polling" procedure hierarchical greedy execution. While this is guaranteed to be no worse than the hierarchically optimal or recursively optimal solution and may be considerably better, it still does not provide any global optimality guarantees. Great care is required while learning with hierarchical greedy execution. Hauskrecht et al. (1998) discuss decomposition and solution techniques that make optimality guarantees, but unfortunately, unless the MDP can be decomposed into very weakly coupled smaller MDPs, the computational complexity is not necessarily reduced. Benefits will still accrue if the options or subtask policies can be reused and amortized over multiple MDPs.

Automatic Decomposition

In the above approaches the programmer is expected to manually decompose the overall problem into a hierarchy of subtasks. Methods to automatically decompose problems include ones that look for subgoal bottleneck or landmark states, and ones that find common behavior trajectories or region policies. For example, in Fig. 1 the agent will exit one of the two starting room doorways on the way to the goal. The states adjacent to each doorway will be visited more frequently in successful trials than other states.

Both NQL (nested Q learning) Digney (1998) and McGovern (2002) use this idea to identify subgoals. Moore et al. (1999) suggest that, for some navigation tasks, performance is insensitive to the position of landmarks and an (automatic) randomly generated set of landmarks does not show widely varying results from more purposefully positioned ones. Hengst has explored automatic learning of MAXQ-like task-hierarchies from the agent's interactive experience with the environment, automatically finding common regions and generating subgoals when the agent's prediction fails. Methods include state abstraction with discounting for infinite horizon problems and decompositions of problems to form partial-order task-hierarchies (Hengst 2008). When there are no cycles in the causal graph the variable influence structure analysis (VISA) algorithm (Jonsson and Barto 2006) performs hierarchical decomposition of factored Markov decision processes using a given dynamic Bayesian network model of actions. Konidaris and Barto (2009) introduce a skill discovery method for reinforcement learning in continuous domains that constructs chains of skills leading to an end-of-task reward.

Given space limitations we cannot adequately cover all the research in hierarchical reinforcement learning, but we trust that the material above will provide a starting point.

Cross-References

- ▶ Associative Reinforcement Learning
- ▶ Average-Reward Reinforcement Learning
- ▶ Bayesian Reinforcement Learning
- ▶ Credit Assignment
- ▶ Markov Decision Processes
- ▶ Model-Based Reinforcement Learning
- ▶ Policy Gradient Methods
- ▶ Q-Learning
- ▶ Reinforcement Learning
- ▶ Relational Reinforcement Learning
- ▶ Structured Induction
- ▶ Temporal Difference Learning

Recommended Reading

Ashby R (1956) Introduction to cybernetics. Chapman & Hall, London

Barto A, Mahadevan S (2003) Recent advances in hiearchical reinforcement learning. Spec Issue Reinf Learn Discret Event Syst J 13:41–77

Dayan P, Hinton GE (1992) Feudal reinforcement learning. In: Advances in neural information processing systems 5 NIPS conference, Denver, 2–5 Dec 1991. Morgan Kaufmann, San Francisco

Dietterich TG (2000) Hierarchical reinforcement learning with the MAXQ value function decomposition. J Artif Intell Res 13:227–303

Digney BL (1998) Learning hierarchical control structures for multiple tasks and changing environments. In: From animals to animats 5: proceedings of the fifth international conference on simulation of adaptive behaviour, SAB 98, Zurich, 17–21 Aug 1998. MIT, Cambridge

Ghavamzadeh M, Mahadevan S (2002) Hierarchically optimal average reward reinforcement learning. In: Sammut C, Achim Hoffmann (eds) Proceedings of the nineteenth international conference on machine learning, Sydney. Morgan-Kaufman, San Francisco, pp 195–202

Hauskrecht M, Meuleau N, Kaelbling LP, Dean T, Boutilier C (1998) Hierarchical solution of Markov decision processes using macro-actions. In: Fourteenth annual conference on uncertainty in artificial intelligence, Madison, pp 220–229

Hengst B (2008) Partial order hierarchical reinforcement learning. In: Australasian conference on artificial intelligence, Auckland, Dec 2008. Springer, Berlin, pp 138–149

Jonsson A, Barto A (2006) Causal graph based decomposition of factored MDPs. J Mach Learn Res 7:2259–2301

Kaelbling LP (1993) Hierarchical learning in stochastic domains: preliminary results. In: Proceedings of the tenth international conference on machine learning. Morgan Kaufmann, San Mateo, pp 167–173

Konidaris G, Barto A (2009) Skill discovery in continuous reinforcement learning domains using skill chaining. In: Bengio Y, Schuurmans D, Lafferty J, Williams CKI, Culotta A (eds) Advances in neural information processing systems 22, Vancouver, pp 1015–1023

McGovern A (2002) Autonomous discovery of abstractions through interaction with an environment. In: SARA. Springer, London, pp 338–339

Moore A, Baird L, Kaelbling LP (1999) Multi-value functions: efficient automatic action hierarchies for multiple goal MDPs. In: Proceedings of the international joint conference on artificial intelligence, Stockholm. Morgan Kaufmann, San Francisco, pp 1316–1323

Parr R, Russell SJ (1997) Reinforcement learning with hierarchies of machines. In: NIPS, Denver

Puterman ML (1994) Markov decision processes: discrete stochastic dynamic programming. New York, Wiley

Ryan MRK, Reid MD (2000) Using ILP to improve planning in hierarchical reinforcement learning. In: Proceedings of the tenth international conference on inductive logic programming, ILP 2000, London. Springer, London

Singh S (1992) Reinforcement learning with a hierarchy of abstract models. In: Proceedings of the tenth national conference on artificial intelligence, San Jose

Sutton RS, Precup D, Singh SP (1999) Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. Artif Intell 112(1–2): 181–211

Watkins CJCH (1989) Learning from delayed rewards. PhD thesis, King's College

Higher-Order Logic

John Lloyd
The Australian National University, Canberra,
ACT, Australia

Definition

Higher-order logic is a logic that admits so-called higher-order functions, which are functions that can have functions as arguments or return a function as a result. The expressive power that comes from higher-order functions makes the logic highly suitable for representing individuals, predicates, features, background theories, and hypotheses and performing the necessary reasoning, in machine learning applications.

Motivation and Background

Machine learning tasks naturally require knowledge representation and reasoning. The individuals that are the subject of learning, the training examples, the features, the background theory, and the hypothesis languages all have to be represented. Furthermore, reasoning, usually in the form of computation, has to be performed.

Logic is a convenient formalism in which knowledge representation and reasoning can be carried out; indeed, it was developed exactly for this purpose. For machine learning applications, quantification over variables is generally needed, so that, at a minimum, ▸ first-order logic should be used. Here, the use of higher-order logic for this task is outlined. Higher-order logic admits higher-order functions that can have functions as arguments or return a function as a result. This means that the expressive power of higher-order logic is greater than first-order logic so that some expressions of higher-order logic are difficult or impossible to state directly in first-order logic. For example, sets can be represented by ▸ predicates which are terms in higher-order logic, and operations on sets can be implemented by higher-order functions. Grammars that generate spaces of predicates can be easily expressed. Also the programming idioms of functional programming languages become available.

The use of higher-order logic in learning applications began around 1990 when researchers argued for the advantages of lifting the concept of ▸ least general generalization in the first-order setting to the higher-order setting (Dietzen and Pfenning 1992; Feng and Muggleton 1992; Lu et al. 1998). A few years later, Muggleton and Page (1994) advocated the use of higher-order concepts, especially sets, for learning applications. Then the advantages of a type system and also higher-order facilities for concept learning were presented in Flach et al. (1998). Higher-order logic is also widely used in other parts of computer science, for example, theoretical computer science, functional programming, and verification of software.

Most treatments of higher-order logic can be traced back to Church's simple theory of types (Church 1940). Recent accounts can be found, for example, in Andrews (2002), Fitting (2002), and Wolfram (1993). For a highly readable account of the advantages of working in higher-order rather than first-order logic, Farmer (2008) is strongly recommended. An account of higher-order logic specifically intended for learning applications is in Lloyd (2003), which contains much more detail about the knowledge representation and reasoning issues that are discussed below.

Theory

Logic

To begin, here is one formulation of the syntax of higher-order logic which gives prominence to a type system that is useful for machine learning applications, in particular.

An *alphabet* consists of four sets: a set $\mathfrak{T}$ of type constructors, a set $\mathfrak{P}$ of parameters, a set $\mathfrak{C}$ of constants, and a set $\mathfrak{V}$ of variables. Each type constructor in $\mathfrak{T}$ has an arity. The set $\mathfrak{T}$ always includes the type constructor Ω of arity 0. Ω is the type of the booleans. Each constant in $\mathfrak{C}$ has a signature (i.e., type declaration). The set $\mathfrak{V}$ is denumerable. Variables are typically denoted by $x, y, z, \ldots$. The parameters are type variables that provide polymorphism in the logic; they are ignored for the moment.

Here is the definition of a type (for the non-polymorphic case).

Definition A *type* is defined inductively as follows:

1. If T is a type constructor of arity k and $\alpha_1, \ldots, \alpha_k$ are types, then $T \ \alpha_1 \ldots \alpha_k$ is a type. (Thus, a type constructor of arity 0 is a type.)
2. If α and β are types, then $\alpha \to \beta$ is a type.

3. If $\alpha_1, \ldots, \alpha_n$ are types, then $\alpha_1 \times \cdots \times \alpha_n$ is a type.

The set $\mathfrak{C}$ always includes the following constants:

1. $\top$ and $\bot$, having signature Ω.
2. $=_\alpha$, having signature $\alpha \to \alpha \to \Omega$, for each type α.
3. $\neg$, having signature $\Omega \to \Omega$.
4. $\wedge$, $\vee$, $\longrightarrow$, $\longleftarrow$, and $\longleftrightarrow$, having signature $\Omega \to \Omega \to \Omega$.
5. Σ_α and Π_α, having signature $(\alpha \to \Omega) \to \Omega$, for each type α.

The intended meaning of $=_\alpha$ is identity (i.e., $=_\alpha x\ y$ is $\top$ if x and y are identical), the intended meaning of $\top$ is true, the intended meaning of $\bot$ is false, and the intended meanings of the connectives $\neg$, $\wedge$, $\vee$, $\longrightarrow$, $\longleftarrow$, and $\longleftrightarrow$ are as usual. The intended meanings of Σ_α and Π_α are that Σ_α maps a predicate to $\top$ if the predicate maps at least one element to $\top$ and Π_α maps a predicate to $\top$ iff the predicate maps all elements to $\top$.

Here is the definition of a term (for the non-polymorphic case).

Definition A *term*, together with its type, is defined inductively as follows:

1. A variable in $\mathfrak{V}$ of type α is a term of type α.
2. A constant in $\mathfrak{C}$ having signature α is a term of type α.
3. If t is a term of type β and x a variable of type α, then $\lambda x.t$ is a term of type $\alpha \to \beta$.
4. If s is a term of type $\alpha \to \beta$ and t a term of type α, then $(s\ t)$ is a term of type β.
5. If $t_1, \ldots, t_n$ are terms of type $\alpha_1, \ldots, \alpha_n$, respectively, then $(t_1, \ldots, t_n)$ is a term of type $\alpha_1 \times \cdots \times \alpha_n$.

A *formula* is a term of type Ω. Terms of the form $(\Sigma_\alpha\ \lambda x.t)$ are written as $\exists_\alpha\, x.t$, and terms of the form $(\Pi_\alpha\ \lambda x.t)$ are written as $\forall_\alpha x.t$ (in accord with the intended meaning of Σ_α and Π_α). Thus, in higher-order logic, each quantifier is obtained as a combination of an abstraction acted on by a suitable function (Σ_α or Π_α).

The polymorphic version of the logic extends what is given above by also having available parameters. The definition of a type as above is then extended to polymorphic types that may contain parameters, and the definition of a term as above is extended to terms that may have polymorphic types.

Reasoning in higher-order logic can consist of theorem proving, via resolution or tableaus, for example, or can consist of equational reasoning, as is embodied in the computational mechanisms of functional programming languages, for example. Theorem proving and equational reasoning can even be combined to produce more flexible reasoning systems. Determining whether a formula is a theorem is, of course, undecidable.

The semantics for higher-order logic is generally based on Henkin (1950) models. Compared with first-order interpretations, the main extra ingredient is that, for each (closed) type of the form $\alpha \to \beta$, there is a domain that consists of some set of functions from the domain corresponding to α to the domain corresponding to β. There exist proof procedures that are sound and complete with respect to this semantics (Andrews 2002; Fitting 2002).

The logic includes the λ-calculus. Thus, the rules of λ-conversion are available:

1. (α-Conversion) $\lambda x.t \succ_\alpha \lambda y.(t\{x/y\})$, if y is not free in t.
2. (β-Reduction) $(\lambda x.s\ t) \succ_\beta s\{x/t\}$.
3. (η-Reduction) $\lambda x.(t\ x) \succ_\eta t$, if x is not free in t.

Here $s\{x/t\}$ denotes the result of replacing free occurrences of x in s by t, where free variable capture is avoided by renaming the relevant bound variables in s.

Higher-order generalization is introduced through the concept of least general generalization as follows (Feng and Muggleton 1992). A term s is *more general* than a term t if there is a substitution θ such that $s\theta$ is λ-convertible to t. A term t is a *common generalization* of a set T of terms if t is more general than each of the terms in T. A term t is a *least general generalization* of a set T of terms if t is a common generalization of T and, for all common generalizations s of T, t is not strictly more general than s.

Knowledge Representation

In machine learning applications, the individuals that are the subject of learning need to be represented. Using logic, individuals are most naturally represented by (closed) terms. In higher-order logic, advantage can be taken of the fact that sets can be identified with predicates (their characteristic functions). Thus, the set $\{1, 2\}$ is the term

$$\lambda x.\text{if } x = 1 \text{ then } \top \text{ else if } x = 2 \text{ then } \top \text{ else } \bot.$$

This idea generalizes to multisets and similar abstractions. For example,

$$\lambda x.\text{if } x = A \text{ then } 42 \text{ else if } x = B \text{ then } 21 \text{ else } 0$$

is the multiset with 42 occurrences of A and 21 occurrences of B (and nothing else). Thus, abstractions of the form

$$\lambda x.\text{if } x = t_1 \text{ then } s_1 \text{ else } \ldots \text{ if}$$
$$x = t_n \text{ then } s_n \text{ else } s_0$$

are adopted to represent (extensional) sets, multisets, and so on.

These considerations motivate the introduction of the class of basic terms that are used to represent individuals (Lloyd 2003). The definition of basic terms is an inductive one consisting of three parts. The first part covers data types such as lists and trees and uses the same constructs for this as are used in functional programming languages. The second part uses abstractions to cover data types such as (finite) sets and multisets, for which the data can be represented by a finite lookup table. The third part covers data types that are product types and therefore allows the representation of tuples. The definition is inductive in the sense that basic terms include lists of sets of tuples, tuples of sets, and so on.

It is common in learning applications to need to generate spaces of predicates. This is because features are typically predicates and logical hypothesis languages contain predicates. Thus, there is a need to specify grammars that can generate spaces of predicates. In addition to first-order approaches based on refinement operators or antecedent description grammars, higher-order logic offers another approach to this task based on the idea of generating predicates by composing certain primitive functions.

Predicate rewrite systems are used to define spaces of standard predicates, where standard predicates are predicates in a particular syntactic form that involves composing certain functions (Lloyd 2003). A predicate rewrite is an expression of the form $p \rightarrowtail q$, where p and q are standard predicates. The predicate p is called the *head* and q is the *body* of the rewrite. A predicate rewrite system is a finite set of predicate rewrites. One should think of a predicate rewrite system as a kind of grammar for generating a particular class of predicates. Roughly speaking, this works as follows. Starting from the weakest predicate *top*, all predicate rewrites that have *top* (of the appropriate type) in the head are selected to make up child predicates that consist of the bodies of these predicate rewrites. Then, for each child predicate and each redex (i.e., subterm selected for expansion) in that predicate, all child predicates are generated by replacing each redex by the body of the predicate rewrite whose head is identical to the redex. This generation of predicates continues to produce the entire space of predicates given by the predicate rewrite system.

Predicate rewrite systems are a convenient mechanism to specify precise control over the space of predicates that is to be generated. Note that predicate rewrite systems depend essentially on the higher-order nature of the logic since standard predicates are obtained by composition of functions and composition is a higher-order function.

Other ingredients of learning problems, such as background theories and training examples, can also be conveniently represented in higher-order logic.

Reasoning

Machine learning applications require that reasoning tasks be carried out, for example, computing the value of some predicate on some individual. Generally, reasoning in (higher-order) logic can be either theorem proving or purely equational reasoning or a combination of both.

A variety of proof systems have been developed for higher-order logic; these include Hilbert-style systems (Andrews 2002) and tableau systems (Fitting 2002).

Purely equational reasoning includes the computational models of functional programming languages and therefore can be usefully thought of as computation. Typical examples of this approach include the declarative programming languages of Curry (Hanus 2006) and Escher (Lloyd 2003) which are extensions of the functional programming language of Haskell (Peyton Jones 2003). For both Curry and Escher, the Haskell computational model is generalized in such a way as to admit the logic programming idioms.

Alternatively, by suitably restricting the fragment of the logic considered and the proof system, computation systems in the form of declarative programming languages can be developed. A prominent example of this approach is the logic programming language λProlog that was introduced in the 1980s (Nadathur and Miller 1998). In λProlog, program statements are higher-order hereditary Harrop formulas, a generalization of the definite ▶ clauses used by ▶ Prolog. The language provides an elegant use of λ-terms as data structures, metaprogramming facilities, universal quantification, and implications in goals, among other features.

Applications

Higher-order logic has been used in a variety of machine learning settings including decision tree learning, kernels, Bayesian networks, and evolutionary computing. Decision tree learning based on the use of higher-order logic as the knowledge representation and reasoning language is presented in Bowers et al. (2000) and further developed in Ng (2005b). Kernels and distances over individuals represented by basic terms are studied in Gärtner et al. (2004). In Gyftodimos and Flach (2005), Bayesian networks over basic terms are defined, and it is shown there how to construct probabilistic classifiers over such networks. In Ng et al. (2008), higher-order logic is used as the setting for studying probabilistic modeling, inference, and learning. An evolutionary approach to learning higher-order concepts is demonstrated in Kennedy and Giraud-Carrier (1999). In addition, the learnability of hypothesis languages expressed in higher-order logic is investigated in Ng (2005a, 2006).

Cross-References

- ▶ First-Order Logic
- ▶ Inductive Logic Programming
- ▶ Learning from Structured Data
- ▶ Propositional Logic

Recommended Reading

Andrews PB (2002) An introduction to mathematical logic and type theory: to truth through proof, 3rd edn. Kluwer Academic, Dordrecht

Bowers AF, Giraud-Carrier C, Lloyd JW (2000) Classification of individuals with complex structure. In: Langley P (ed) Machine learning: proceedings of the seventeenth international conference (ICML 2000), Stanford. Morgan Kaufmann, Stanford, pp 81–88

Church A (1940) A formulation of the simple theory of types. J Symb Log 5:56–68

Dietzen S, Pfenning F (1992) Higher-order and modal logic as a framework for explanation-based generalization. Mach Learn 9:23–55

Farmer W (2008) The seven virtues of simple type theory. J Appl Log 6(3):267–286

Feng C, Muggleton SH (1992) Towards inductive generalisation in higher order logic. In: Sleeman D, Edwards P (eds) Proceedings of the ninth international workshop on machine learning. Morgan Kaufmann, San Mateo, pp 154–162

Fitting M (2002) Types, tableaus, and Gödel's god. Kluwer Academic, Dordrecht

Flach P, Giraud-Carrier C, Lloyd JW (1998) Strongly typed inductive concept learning. In: Page D (ed) Inductive logic programming, 8th international conference, ILP-98, Madison. Lecture notes in artificial intelligence, vol 1446. Springer, Berlin, pp 185–194

Gärtner T, Lloyd JW, Flach P (2004) Kernels and distances for structured data. Mach Learn 57(3):205–232

Gyftodimos E, Flach P (2005) Combining Bayesian networks with higher-order data representations. In: Proceedings of 6th international symposium on intelligent data analysis (IDA 2005), Madrid. Lecture notes in computer science, vol 3646. Springer, Berlin, pp 145–156

Hanus M (ed) (2006) Curry: an integrated functional logic language. http://www.informatik.uni-kiel.de/~curry. Retrieved 21 Dec 2009

Henkin L (1950) Completeness in the theory of types. J Symb Log 15(2):81–91

Kennedy CJ, Giraud-Carrier C (1999) An evolutionary approach to concept learning with structured data. In: Proceedings of the fourth international conference on artificial neural networks and genetic algorithms (ICANNGA'99). Springer, Berlin, pp 331–366

Lloyd JW (2003) Logic for learning. Cognitive technologies. Springer, Berlin

Lu J, Harao M, Hagiya M (1998) Higher order generalization. In: JELIA '98: proceedings of the European workshop on logics in artificial intelligence, Dagstuhl. Lecture notes in artificial intelligence, vol 1489. Springer, Berlin, pp 368–381

Muggleton S, Page CD (1994) Beyond first-order learning: inductive learning with higher-order logic. Technical report PRG-TR-13-94, Oxford University Computing Laboratory

Nadathur G, Miller DA (1998) Higher-order logic programming. In: Gabbay DM, Hogger CJ, Robinson JA (eds) The handbook of logic in artificial intelligence and logic programming, vol 5. Oxford University Press, Oxford, pp 499–590

Ng KS (2005a) Generalization behaviour of alkemic decision trees. In: Inductive logic programming, 15th international conference (ILP 2005), Bonn. Lecture notes in artificial intelligence, vol 3625. Springer, Berlin, pp 246–263

Ng KS (2005b) Learning comprehensible theories from structured data. PhD thesis, Computer Sciences Laboratory, The Australian National University

Ng KS (2006) (Agnostic) PAC learning concepts in higher-order logic. In: European conference on machine learning (ECML 2006), Berlin. Lecture notes in artificial intelligence, vol 4212. Springer, Berlin, pp 711–718

Ng KS, Lloyd JW, Uther WTB (2008) Probabilistic modelling, inference and learning using logical theories. Ann Math Artif Intell 54:159–205. doi:10.1007/s 10472-009-9136-7

Peyton Jones S (ed) (2003) Haskell 98 language and libraries: the revised report. Cambridge University Press, Cambridge

Wolfram DA (1993) The clausal theory of types. Cambridge University Press, Cambridge

Hold-One-Out Error

▶ Leave-One-Out Error

Holdout Data

▶ Holdout Set

Holdout Evaluation

Definition

Holdout evaluation is an approach to ▶ out-of-sample evaluation whereby the available data are partitioned into a ▶ training set and a ▶ test set. The test set is thus ▶ out-of-sample data and is sometimes called the *holdout set* or *holdout data*. The purpose of holdout evaluation is to test a model on different data to that from which it is learned. This provides less biased estimate of learning performance than ▶ in-sample evaluation.

In *repeated holdout evaluation*, repeated holdout evaluation experiments are performed, each time with a different partition of the data, to create a distribution of training and ▶ test sets with which an algorithm is assessed.

Cross-References

▶ Algorithm Evaluation

Holdout Set

Synonyms

Holdout data

Definition

A holdout set is a ▶ data set containing data that are not used for learning and that are used for ▶ evaluation by a learning system.

Cross-References

▶ Evaluation Set
▶ Holdout Evaluation

Hopfield Network

Risto Miikkulainen
Department of Computer Science,
The University of Texas at Austin, Austin,
TX, USA

Synonyms

Recurrent associative memory

Definition

The Hopfield network is a binary, fully recurrent network that, when started on a random activation state, settles the activation over time into a state that represents a solution (Hopfield and Tank 1986). This architecture has been analyzed thoroughly using tools from statistical physics. In particular, with symmetric weights, no self-connections, and asynchronous neuron activation updates, a Lyapunov function exists for the network, which means that the network activity will eventually settle. The Hopfield network can be used as an associate memory or as a general optimizer. When used as an associative memory, the weight values are computed from the set of patterns to be stored. During retrieval, part of the pattern to be retrieved is activated, and the network settles into the complete pattern. When used as an optimizer, the function to be optimized is mapped into the Lyapunov function of the network, which is then solved for the weight values. The network then settles to a state that represents the solution. The basic Hopfield architecture can be extended in many ways, including continuous neuron activations. However, it has limited practical value mostly because it is not strong in either of the above task: as an associative memory, its capacity is approximately $0.15N$ in practice (where N is the number of neurons), and as an optimizer, it often settles into local optima instead of the global one. The ▶ Boltzmann machine extends the architecture with hidden neurons, allowing for better performance in both tasks. However, the Hopfield network has had a large impact in the field because the theoretical techniques developed for it have inspired theoretical approaches for other architectures as well, especially for those of self-organizing systems (e.g., ▶ self-organizing maps, ▶ adaptive resonance theory).

Recommended Reading

Hopfield JJ, Tank DW (1986) Computing with neural circuits: a model. Science 233:624–633

Hyperparameter Optimization

▶ Metalearning

Hypothesis Language

Hendrik Blockeel
Katholieke Universiteit Leuven, Heverlee,
Leuven, Belgium
Leiden Institute of Advanced Computer Science,
Heverlee, Belgium

Synonyms

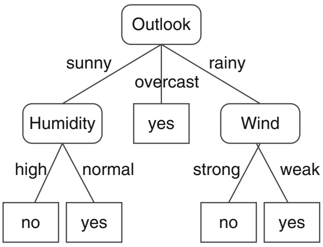
Representation language

Definition

The *hypothesis language* used by a machine learning system is the language in which the hypotheses (also referred to as patterns or models) it outputs are described.

Motivation and Background

Most machine learning algorithms can be seen as a procedure for deriving one or more hypotheses from a set of observations. Both the input (the observations) and the output (the hypotheses) need to be described in some particular language. This language is respectively called the ▶ Observation Language or the hypothesis



IF Outlook=sunny AND Humidity=high THEN Play=no
IF Outlook=sunny AND Humidity=normal THEN Play=yes
IF Outlook=overcast THEN Play=yes
IF Outlook=rainy AND Wind=strong THEN Play=no
IF Outlook=rainy AND Wind=weak THEN Play=yes

Hypothesis Language, Fig. 1 A decision tree and an equivalent rule set

language. These terms are mostly used in the context of symbolic learning, where these languages are often more complex than in subsymbolic or statistical learning. For instance, hypothesis languages have received a lot of attention in the field of ▸ Inductive Logic Programming, where systems typically take as one of their input parameters a declarative specification of the hypothesis language they are supposed to use (which is typically a strict subset of full clausal logic). Such a specification is also called a ▸ Language Bias.

Examples of Hypothesis Languages

The hypothesis language used obviously depends on the learning task that is performed. For instance, in predictive learning, the output is typically a function, and thus the hypothesis language must be able to represent functions; whereas in clustering the language must have constructs for representing clusters (sets of points). Even for one and the same goal, different languages may be used; for instance, decision trees and rule sets can typically represent the same type of functions, so the difference between these two is mostly syntactic.

In the following section, we discuss briefly a few different formalisms for representing hypotheses. For most of these, there are separate entries in this volume that offer more detail on the specifics of that formalism.

Decision Trees and Rule Sets

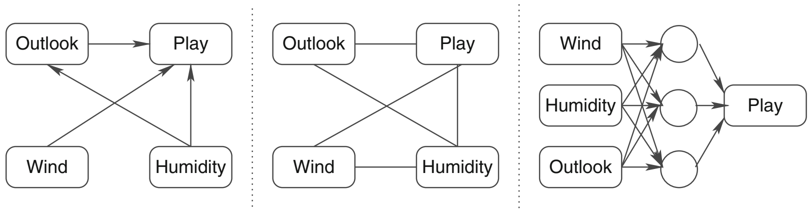
A ▸ Decision Tree represents a decision process where consecutive tests are performed on an instance to determine the value of its target variable, and at each step in this process, the test that is performed depends on the outcome of previous tests. Each leaf of the tree contains the set of all instances that fulfill the conjunctions of all conditions on the path from the root to this leaf, and as such a tree can easily be written as a set of if-then rules where each rule contains one such conjunction. If the target variable is boolean, this format corresponds to disjunctive normal form.

Figure 1 shows a decision tree and the corresponding rule set. (Inspired by Mitchell 1997).

Graphical Models

The term “graphical models” usually refers to probabilistic models where the joint distribution over a set of variables is defined as the product of a number of joint distributions over subsets of these variables (i.e., a factorization), and this factorization is defined by a graph structure. The graph may be directed, in which case we speak of a ▸ Bayesian Network, undirected, in which case we speak of a ▸ Markov Network, or even a mix of the two (so-called chain graphs). In a Bayesian network, the constituent distributions of the factorization are conditional probability functions associated with each node. In a Markov network, the constituent distributions are potential functions associated with each clique in the graph.

Two learning settings can be distinguished: learning the parameters of a graphical model given the model structure (the graph), and learning both structure and parameters of the model. In the first case, the graph is in fact a language bias specification: the user forces the learner to return a hypothesis that lies within the set of hypotheses



Hypothesis Language, Fig. 2 A Bayesian network, a Markov network, and a neural network

representable by this particular structure. In the second case, the structure of the graph makes explicit certain independencies that are hypothesized to exist between the variables (thus it is part of the hypothesis itself).

Figure 2 shows examples of possible graphical models that might be learned from data. For details about the interpretation of such graphical models, we refer to the respective entries in this encyclopedia.

Neural Networks

► Neural Networks are typically used to represent complex nonlinear functions. A neural network can be seen as a directed graph where the nodes are variables and edges indicate which variables depend on which other variables. Some nodes represent the observed input variables x_i and output variables y, and some represent new variables introduced by the network. Typically, a variable depends, in a nonlinear way, on a linear combination of those variables that directly precede it in the directed graph. The parameters of the network are numerical edge labels that represent the weight of a parent variable in that linear combination.

As with graphical models, one can learn the parameters of a neural network with a given structure, in which case the structure serves as a language bias; or one can learn both the structure and the parameters of the network.

Figure 2 shows an example of a neural network. We refer to the respective entry for more information on neural networks.

Instance-Based Learning

In the most basic version of ► instance-based learning, the training data set itself represents the hypothesis. As such, the hypothesis language is simply the powerset of the observation language. Because many instance-based learners rescale the dimensions of the input space, the vector containing the rescaling factors can be seen as part of the hypothesis. Similarly, some methods derived from instance-based learning build a model in which the training set instances are replaced by prototypes (one prototype being representative for a set of instances) or continuous functions approximating the instances.

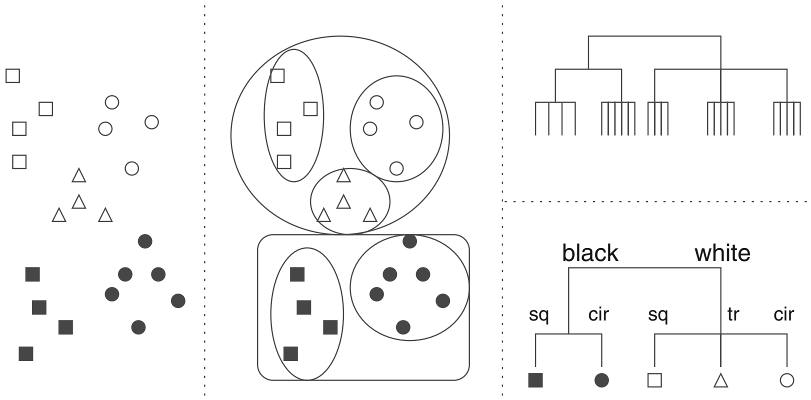
H

Clustering

In clustering tasks, there is an underlying assumption that there is a certain structure in the data set; that is, the data set is really a mixture of elements from different groups or clusters, with each cluster corresponding to a different population. The goal is to describe these clusters or populations and to indicate which data elements belong to which cluster.

Some clustering methods define the clusters extensionally, that is, they describe the different clusters in the dataset by just enumerating the elements in the dataset that belong to them. Other methods add an intensional description to the clusters, defining the properties that an instance should have in order to belong to the cluster; as such, these intensional methods attempt to describe the population that the cluster is a sample from. Some methods recursively group the clusters into larger clusters, building a cluster hierarchy. Figure 3 shows an example of such a cluster hierarchy.

The term “mixture models” typically refers to methods that return a probabilistic model (e.g., a Gaussian distribution with specified parameters) for each separate population identified. Being



Hypothesis Language, Fig. 3 A hierarchical clustering: *left*, the data set; *middle*: an extensional clustering shown on the data set; *right*, above: the corresponding extensional clustering tree; *right*, below: a corresponding intensional clustering tree, where the clusters are described based on color and shape of their elements

probabilistic in nature, these methods typically also assign data elements to the populations in a probabilistic, as opposed to deterministic, manner.

First-Order Logic Versus Propositional Languages

In symbolic machine learning, a distinction is often made between the so-called attribute-value (or propositional) and relational (or first-order) languages. The terminology "propositional" versus "first-order" originates in logic. In ▶ Propositional Logic, only the existence of propositions, which can be true or false, is assumed, and these propositions can be combined with the usual logical connectives into logical formulae. In ▶ First-Order Predicate Logic, the existence of a universe of objects is assumed as well as the existence of predicates that can express certain properties of and relationships between these objects. By adding variables and quantifiers, one can describe deductive reasoning processes in first-order logic that cannot be described in propositional logic. For instance, in propositional logic, one could state propositions *Socrates_is_human* and *all_humans_are_mortal* (both are statements that may be true or false), but there is no inherent relationship between them. In first order logic, the formulae $human(Socrates)$ and $\forall x\colon human(x) \rightarrow mortal(x)$ allow one to deduce $mortal(Socrates)$. A more extensive explanation of the differences between propositional and first-order logic can be found in the entry on ▶ First-Order Logic.

Many machine learning approaches use an essentially propositional language for describing observations and hypotheses. In the fields of Inductive Logic Programming and ▶ Relational Learning, more powerful languages are used, with an expressiveness closer to that of first-order logic. Many of the representation languages mentioned above, which are essentially propositional, have been extended towards the first-order logic context.

The simplest example is that of rule sets. If-then rules have a straightforward counterpart in first-order logic in the form of ▶ Clauses, which are usually written as logical implications where all variables are interpreted as universally quantified. For instance, the rule "IF Human = true THEN Mortal = true" can be written in clausal form as

$$\mathrm{mortal}(x) \leftarrow \mathrm{human}(x). \tag{1}$$

Propositional rules correspond to clauses that refer to only one object (and the object reference is implicit). A rule such as

$$\mathrm{grandparent}(x, y) \leftarrow \mathrm{parent}(x, z), \mathrm{parent}(z, y) \tag{2}$$

(expressing that, for any x, y, z, whenever x is a parent of z and z is a parent of y, x is a grandparent of y) has no translation into propositional logic that retains the inference capacity of the first-order logic clause.

Clauses are a natural first-order logic equivalent to the if-then rules typically returned by rule learners, and many of the other representation languages have also been upgraded to the relational or first-order-logic context. For instance, several researchers (e.g., Blockeel and De Raedt 1998) have upgraded the formalism of decision trees toward "structural" or "first-order logic" decision trees. Probabilistic relational models (Getoor et al. 2001) and Bayesian logic programs (Kersting and De Raedt 2001) are examples of how Bayesian networks have been upgraded, while Markov networks have been lifted to "Markov logic" (Richardson and Domingos 2006).

Further Reading

Most of the literature on hypothesis and observation languages is found in the area of inductive logic programming. Excellent starting points, containing extensive examples of bias specifications, are *Relational Data Mining* by Džeroski and Lavra (2001), *Logic for Learning* by Lloyd (2003), and *Logical and Relational Learning* by De Raedt (2008).

De Raedt (1998) compares a number of different observation and hypothesis languages with respect to their expressiveness, and indicates relationships between them.

Cross-References

▶ First-Order Logic
▶ Hypothesis Space
▶ Inductive Logic Programming
▶ Observation Language

Recommended Reading

Blockeel H, De Raedt L (1998) Top-down induction of first order logical decision trees. Artif Intell 101(1–2):285–297

De Raedt L (1998) Attribute-value learning versus inductive logic programming: the missing links (extended abstract). In: Page D (ed) Proceedings of the eighth international conference on inductive logic programming. Lecture notes in artificial intelligence, vol 1446. Springer, Berlin, pp 1–8

De Raedt L (2008) Logical and relational learning. Springer, Berlin

Džeroski S, Lavraè N (ed) (2001) Relational data mining. Springer, Berlin

Getoor L, Friedman N, Koller D, Pfeffer A (2001) Learning probabilistic relational models. In: Dzeroski S, Lavrac N (eds) Relational data mining. Springer, Berlin, pp 307–334

Kersting K, De Raedt L (2001) Towards combining inductive logic programming and Bayesian networks. In: Rouveirol C, Sebag M (eds) Proceedings of the 11th international conference on inductive logic programming. Lecture notes in computer science, vol 2157. Springer, Berlin, pp 118–131

Lloyd JW (2003) Logic for learning. Springer, Berlin

Mitchell T (1997) Machine learning. McGraw Hill, New York

Richardson M, Domingos P (2006) Markov logic networks. Mach Learn 62(1–2):107–136

Hypothesis Space

Hendrik Blockeel
Katholieke Universiteit Leuven, Heverlee, Leuven, Belgium
Leiden Institute of Advanced Computer Science, Heverlee, Belgium

Synonyms

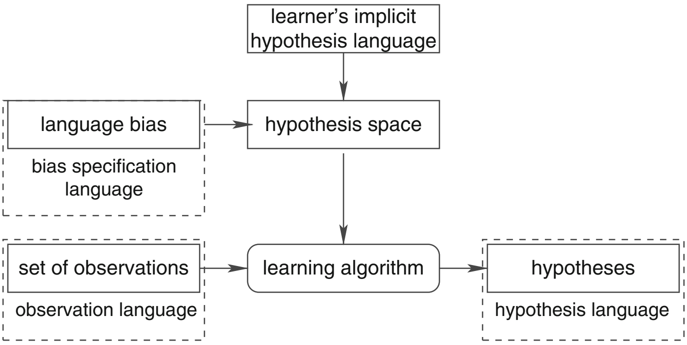
Model space

Definition

The *hypothesis space* used by a machine learning system is the set of all hypotheses that might possibly be returned by it. It is typically defined by a ▶ Hypothesis Language, possibly in conjunction with a ▶ Language Bias.

Motivation and Background

Many machine learning algorithms rely on some kind of search procedure: given a set of observations and a space of all possible hypotheses that



Hypothesis Space, Fig. 1 Structure of learning systems that derive one or more hypotheses from a set of observations

might be considered (the "hypothesis space"), they look in this space for those hypotheses that best fit the data (or are optimal with respect to some other quality criterion).

To describe the context of a learning system in more detail, we introduce the following terminology. The key terms have separate entries in this encyclopedia, and we refer to those entries for more detailed definitions.

A learner takes observations as inputs. The ▸ Observation Language is the language used to describe these observations.

The hypotheses that a learner may produce, will be formulated in a language that is called the Hypothesis Language. The *hypothesis space* is the set of hypotheses that can be described using this hypothesis language.

Often, a learner has an implicit, built-in, hypothesis language, but in addition the set of hypotheses that can be produced can be restricted further by the user by specifying a language bias. This language bias defines a subset of the hypothesis language, and correspondingly a subset of the hypothesis space. A separate language, called the ▸ Bias Specification Language, is used to define this language bias. Note that while elements of a hypothesis language refer to a single hypothesis, elements of a bias specification language refer to sets of hypotheses, so these languages are typically quite different. Bias specification languages have been studied in detail in the field of ▸ Inductive Logic Programming.

The terms "hypothesis language" and "hypothesis space" are sometimes used in the broad sense (the language that the learner is inherently restricted to, e.g., Horn clauses), and sometimes in a more narrow sense, referring to the smaller language or space defined by the language bias.

The structure of a learner, in terms of the above terminology, is summarized in Fig. 1.

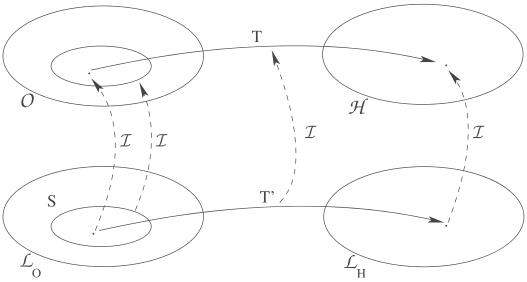
Theory

For a given learning problem, let us denote with $\mathcal{O}$ the set of all possible observations (sometimes also called the instance space), and with $\mathcal{H}$ the hypothesis space, i.e., the set of all possible hypotheses that might be learned. Let $2X$ denote the power set of a set X. Most learners can then be described abstractly as a function $T : 2^{\mathcal{O}} \rightarrow \mathcal{H}$, which takes as input a set of observations (also called the training set) $S \subseteq \mathcal{O}$, and produces as output a hypothesis $h\epsilon\mathcal{H}$.

In practice, the observations and hypotheses are represented by elements of the observation language $\mathcal{L}_O$ and the hypothesis language $\mathcal{L}_H$, respectively. The connection between language elements and what they represent is defined by functions $\mathcal{I}_O : \mathcal{L}_O \rightarrow \mathcal{O}$ (for observations) and $\mathcal{I}_H : \mathcal{L}_H \rightarrow \mathcal{H}$ (for hypotheses). This mapping is often, but not always, bijective. When it is not bijective, different representations for the same hypothesis may exist, possibly leading to redundancy in the learning process.

We will use the symbol $\mathcal{I}$ as a shorthand for $\mathcal{I}_O$ or $\mathcal{I}_H$. We also define the application of $\mathcal{I}$ to any set S as $\mathcal{I}_S = \{\mathcal{I}(x)|x \in S\}$, and to any function f as $\mathcal{I}(f) = g \Leftrightarrow \forall x : g(\mathcal{I}(x)) = \mathcal{I}(f(x))$.

Thus, a machine learning system really implements a function $T' : 2^{\mathcal{L}_O} \rightarrow \mathcal{L}_H$, rather than a function $T : 2^{\mathcal{O}} \rightarrow \mathcal{H}$. The connection



Hypothesis Space, Fig. 2 Illustration of the interpretation function $\mathcal{I}$ mapping $\mathcal{L}_O, \mathcal{L}_H$, and T' onto $\mathcal{O}, \mathcal{H}$, and T

between T' and T is straightforward: for any $S \subseteq \mathcal{L}_O$ and $h \in \mathcal{L}_H, T'(S) = h$ if and only if $T(\mathcal{I}(S)) = \mathcal{I}(h)$; in other words: $T = \mathcal{I}(T')$.

Figure 2 summarizes these languages and spaces and the connections between them. We further illustrate them with a few examples.

Example 1 In supervised learning, the observations are usually pairs (x, y) with $x \in X$ an instance and $y \in Y$ its label, and the hypotheses are functions mapping X onto Y. Thus $\mathcal{O} = X \times Y$ and $\mathcal{H} \subseteq Y^X$, with Y^X the set of all functions from X to Y. $\mathcal{L}_O$ is typically chosen such that $\mathcal{I}(\mathcal{L}_O) = \mathcal{O}$, i.e., each possible observation can be represented in $\mathcal{L}_O$. In contrast to this, in many cases $\mathcal{I}(\mathcal{L}_H)$ will be a strict subset of Y^X, i.e., $\mathcal{I}(\mathcal{L}_H) \subset Y^X$. For instance, $\mathcal{L}_H$ may contain representations of all polynomial functions from X to Y if $X = \mathbf{R}^n$ and $Y = \mathbf{R}$ (with $\mathbf{R}$ the set of real numbers), or may be able to represent all conjunctive concepts over X when $X = \mathbf{B}^n$ and $Y = \mathbf{B}$ (with $\mathbf{B}$ the set of booleans).

When $\mathcal{I}(\mathcal{L}_H \subset Y^X$, the learner cannot learn every imaginable function. Thus, $\mathcal{L}_H$ reflects an inductive bias that the learner has, called its *language bias*. We can distinguish an implicit language bias, inherent to the learning system, and corresponding to the hypothesis language (space) in the broad sense, and an explicit language bias formulated by the user, corresponding to the hypothesis language (space) in the narrow sense.

Example 2 Decision tree learners and rule set learners use a different language for representing the functions they learn (call these languages $\mathcal{L}_{DT}$ and $\mathcal{L}_{RS}$, respectively), but their language bias is essentially the same: for instance, if $X = \mathbf{B}^n$ and $Y = \mathbf{B}, \mathcal{I}(\mathcal{L}_{DT}) = \mathcal{I}(\mathcal{L}_{RS}) = Y^X$: both trees and rule sets can represent any boolean function from $\mathbf{B}^n$ to $\mathbf{B}$.

In practice a decision tree learner may employ constraints on the trees that it learns, for instance, it might be restricted to learning trees where each leaf contains at least two training set instances. In this case, the actual hypothesis language used by the tree learner is a subset of the language of all decision trees.

Generally, if the hypothesis language in the broad sense is $\mathcal{L}_H$ and the hypothesis language in the narrow sense is $\mathcal{L}'_H$, then we have $\mathcal{L}'_H \subseteq \mathcal{L}_H$ and the corresponding spaces fulfill (in the case of supervised learning)

$$\mathcal{I}(\mathcal{L}'_H) \subseteq \mathcal{I}(\mathcal{L}_H) \subseteq Y^X. \qquad (1)$$

Clearly, the choice of $\mathcal{L}_O$ and $\mathcal{L}_H$ determines the kind of patterns or hypotheses that can be expressed. See the entries on Observation Language and Hypothesis Language for more details on this.

Further Reading

The term "hypothesis space" is ubiquitous in the machine learning literature, but few articles discuss the concept itself. In Inductive Logic Programming, a significant body of work exists on how to define a language bias (and thus a hypothesis space), and on how to automatically weaken the bias (enlarge the hypothesis space) when a given bias turns out to be too strong. The expressiveness of particular types of learners (e.g., classes of ▶ Neural Networks) has been studied, and this relates directly to the hypothesis space they use. We refer to the respective entries in this volume for more information on these topics.

Cross-References

▶ Bias Specification Language
▶ Hypothesis Language
▶ Inductive Logic Programming
▶ Observation Language

Recommended Reading

De Raedt L (1992) Interactive theory revision: an inductive logic programming approach. Academic, London

Nédellec C, Adé H, Bergadano F, Tausend B (1996) Declarative bias in ILP. In: De Raedt L (ed) Advances in inductive logic programming. Frontiers in artificial intelligence and applications, vol 32. IOS Press, Amsterdam, pp 82–103

I

Identification

▶ Classification

Identity Uncertainty

▶ Entity Resolution
▶ Record Linkage

Idiot's Bayes

▶ Naïve Bayes

Immune Computing

▶ Artificial Immune Systems

Immune Network

A proposed theory that the immune system is capable of achieving immunological memory by the existence of a mutually reinforcing network of B-cells. This network of B-cells forms due to the ability of the paratopes, located on B-cells, to match against the idiotopes on other B-cells. The binding between the idiotopes and paratopes has the effect of stimulating the B-cells. This is because the paratopes on B-cells react to the idiotopes on similar B-cells, as it would an antigen. However, to counter the reaction there is a certain amount of suppression between the B-cells which acts as a regulatory mechanism. This interaction of the B-cells due to the network was said to contribute to a *stable* memory structure and account for the retainment of memory cells, even in the absence of antigen. This interaction of cells forms the basis of inspiration for a large number of AIS algorithms, for example aiNET.

Immune-Inspired Computing

▶ Artificial Immune Systems

Immunocomputing

▶ Artificial Immune Systems

Immunological Computation

▶ Artificial Immune Systems

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

Implication

▸ Entailment

Improvement Curve

▸ Learning Curves in Machine Learning

Incremental Learning

Paul E. Utgoff
University of Massachusetts, Amherst, MA, USA

Definition

Incremental learning refers to any ▸ online learning process that learns the same model as would be learned by a ▸ batch learning algorithm.

Motivation and Background

Incremental learning is useful when the input to a learning process occurs as a stream of distinct observations spread out over time, with the need or desire to be able to use the result of learning at any point in time, based on the input observations received so far. In principle, the stream of observations may be infinitely long, or the next observation long delayed, precluding any hope of waiting until all the observations have been received. Without the ability to forestall learning, one must commit to a sequence of hypotheses or other learned artifacts based on the inputs observed up to the present. One would rather not simply accumulate and store all the inputs and, upon receipt of each new one, apply a batch learning algorithm to the entire sequence of inputs received so far. It would be preferable computationally if the existing hypothesis or other artifact of learning could be updated in response to each newly received input observation.

Theory

Consider the problem of computing the balance in one's checkbook account. Most would say that this does not involve learning, but it illustrates an important point about incremental algorithms. One procedure, a batch algorithm based on the fundamental definition of balance, is to compute the balance as the sum of the deposits less the sum of the checks and fees. As deposit, check, and fee transactions accumulate, this definition remains valid. There is an expectation that there will be more transactions in the future, and there is also a need to compute the balance periodically to ensure that no contemplated check or fee will cause the account to become overdrawn. We cannot wait to receive all of the transactions and then compute the balance just once.

One would prefer an incremental algorithm for this application, to reduce the cost of computing the balance after each transaction. This can be accomplished by recording and maintaining one additional piece of information, the balance after the nth transaction. It is a simple matter to prove that the balance after n transactions added to the amount of transaction $n + 1$ provides the balance after $n + 1$ transactions. This is because the sums of the fundamental definition for $n + 1$ transactions can be rewritten as the sums of the fundamental definition for n transactions plus the amount of the nth transaction. This incremental algorithm reduces the computation necessary to know the balance after each transaction, but it increases the bookkeeping effort somewhat due to the need for an additional variable.

Now consider the problem of learning the mean of a real-valued variable from a stream of observed values of this variable. Though simple, most would say that this does involve learning, because one estimates the mean from observations, without ever establishing the mean definitively. The fundamental definition for the mean requires summing the observed values and then dividing by the number of observed values. As each new observation is received, one could compute the new mean. However, one can reduce the computational cost by employing an incremental algorithm. For n observations, we could just as well have observed exactly the n occurrences of the mean. The sum of these observations divided by n would produce the mean. If we were to be provided with an $n + 1$ observation, we could compute the new sum of the $n + 1$ observations as n cases of the mean value plus the new observation, divided by $n + 1$. This reduces the cost of computing the mean after each observation to one multiplication, two addition, and one division operations. There is a small increase in bookkeeping in maintaining the counter n of how many observations have been received and the mean m after n observations.

In both of the above examples, the need to record the fundamental data is eliminated. Only a succinct summary of the data needs to be retained. For the checkbook balance, only the balance after n transactions needs to be stored, making the specific amounts for the individual transactions superfluous. For the mean of a variable, only the mean m after n observations and the number n of observations need to be retained, making the specific values of the individual observations superfluous. Due to this characteristic, incremental algorithms are often characterized as memoryless, not because no memory at all is required but because no memory of the original data items is needed. An incremental algorithm is not required to be memoryless, but the incremental algorithm must operate by modifying its existing knowledge, not by hiding the application of the corresponding batch algorithm to the accumulated set of observations. The critical issue is the extent to which computation is reduced compared to starting with all the data observations and nothing more. An essential aspect for an incremental algorithm is that the obtained result be identical to that indicated by the fundamental definition of the computation to be performed.

A point of occasional confusion is whether to call an algorithm incremental when it makes adjustments to its data structures in response to a new data observation. The answer depends on whether the result is the same that one would obtain when starting with all the observations in hand. If the answer is no, then one may have an online learning algorithm that is not an incremental learning algorithm. For example, consider two alternative formulations of the problem mentioned above of learning the mean of a variable. Suppose that the count of observations, held in the variable n, is not permitted to exceed some constant, say 100. Then the mean after n observations coupled with the minimum of n and 100 no longer summarizes all n observations accurately. Consider a second reformulation. Suppose that the most recent 100 observations are held in a queue. When a new observation is received, it replaces the oldest of the 100 observations. Now the algorithm can maintain a moving average, but not the overall overage. These may be desirable, if one wishes to remain responsive to drift in the observations, but that is another matter. The algorithm would not be considered incremental because it does not produce the same result for all n observations that the corresponding batch algorithm would for these same n observations. The algorithm would be online, and it would be memoryless, but it would not be computing the same learned artifact as the batch algorithm.

These two latter reformulations raise the issue of whether the order in which the observations are received is relevant. It is often possible to determine this by looking at the fundamental definition of the computation to be performed. If the operator that aggregates the observations is commutative, then order is not important. For the checking account balance example above, the fundamental aggregation is accomplished in the summations, and addition is commutative, so the order of the transactions is not relevant to the resulting balance. If a fundamental algorithm operates on a set of observations, then aggregation of a new observation into a set of observations is accomplished by the set union operator, which is commutative. Can one have an incremental algorithm for which order of the observations is important? In principle, yes, provided that the result of the incremental algorithm after observation n is the same as that of the fundamental algorithm for the first n observations.

A final seeming concern for an incremental learning algorithm is whether the selection of future observations ($n+1$ and beyond) is influenced by the first n observations. This is a red herring, because for the n observations, the question of whether the learning based on these observations can be accomplished by a batch algorithm or a corresponding incremental algorithm remains. Of course, if one needs to use the result of learning on the first k instances to help select the $k + 1$ instance, then it would be good sense to choose an incremental learning algorithm. One would rather not apply a batch algorithm to each and every prefix of the input stream. This would require saving the input stream and it would require doing much more computation than is necessary.

We can consider a few learning scenarios which suit incremental learning. An ▸ active learner uses its current knowledge to select the next observation. For a learner that is inducing a classifier, the observation would be an unclassified instance. The active learner selects an unclassified instance, which is passed to an oracle that attaches a correct class label. Then the oracle returns the labeled instance as the next observation for the learner. The input sequence is no longer one of instances for which each was drawn independently according to a probability distribution over the possible instances. Instead, the distribution is conditionally dependent on what the learner currently believes. The learning problem is sequential in its nature. The observation can be delivered in sequence, and an incremental learning algorithm can modify its hypothesis accordingly. For the n observations received so far, one could apply a corresponding batch algorithm, but this would be unduly awkward.

▸ Reinforcement learning is a kind of online learning in which an agent makes repeated trials in a simulated or abstracted world in order to learn a good, or sometimes optimal, policy that maps states to actions. The learning artifact is typically a function V over states or a function Q over state-action pairs. As the agent moves from state to state, it can improve its function over time. The choice of action depends on the current V or Q and on the reward or punishment received at each step. Thus, the sequence of observations consists of state-reward pairs or state-action-reward triples. As with active learning, the sequence of observations can be seen as being conditionally dependent on what the learner currently believes at each step. The function V or Q can be modified after each observation, without retaining the observation. When the function is approximated in an unbiased manner, by using a lookup table for discrete points in the function domain, there is an analogy with the problem of computing a checkbook balance, as described above. For each cell of the lookup table, its value is its initial value plus the sum of the changes, analogously for transactions. One can compute the function value by computing this sum, or one can store the sum in the cell, as the net value of all the changes. An incremental algorithm is preferable both for reasons of time and space.

A k-nearest classifier (see ▸ Instance-Based Learning) is defined by a set of training instances, the observations, and a distance metric that returns the numeric distance between any two instances. The difference between the batch algorithm and the incremental algorithm is slight. The batch algorithm accepts all the observations at once, and the incremental algorithm simply adds each new observation to the set of observations. If, however, there were data structures kept in the background to speed computation, one could distinguish between building those data structures once (batch) and updating those data structures (incremental). One complaint might be that all of the observations are retained. However, these observations do not need to be revisited when a new one arrives. There is an impact on space, but not on time.

A ▸ decision tree classifier may be correct for the n observations observed so far. When the $n+1$ observation is received, an incremental algorithm will restructure the tree as necessary to produce the tree that the batch algorithm would have built for these $n + 1$ observations. To do this, it may be that no restructuring is required at all or that restructuring is needed only in a subtree. This is a case in which memory is required for saving observations in the event that some of them may be needed to be reconsidered from time to time. There is a great savings in time over running the corresponding batch algorithm repeatedly.

Applications

Incremental learning is pervasive, and one can find any number of applications described in the literature and on the web. This is likely due to the fact that incremental learning offers computational savings in both time and space. It is also likely due to the fact that human and animal learning takes place over time. There are sound reasons for incremental learning being essential to development.

Future Directions

Increasingly, machine learning is confronted with the problem of learning from input streams that contain many millions, or more, of observations. Indeed, the stream may produce millions of observations per day. Streams with this many instances need to be handled by methods whose memory requirements do not grow much or at all. Memoryless online algorithms are being developed that are capable of handling this much throughput. Consider transaction streams, say of a telephone company, or a credit card company, or a stock exchange, or a surveillance camera, or eye-tracking data, or mouse movement data. For such a rich input stream, one could sample it, thereby reducing it to a smaller stream. Or, one could maintain a window of observations, giving a finite sample that changes but does not grow over time. There is no shortage of applications that can produce rich input streams. New methods capable of handling such heavy streams have already appeared, and we can expect to see growth in this area.

Cross-References

- ▶ Active Learning
- ▶ Cumulative Learning
- ▶ Online Learning

Recommended Reading

Domingos P, Hulten G (2003) A general framework for mining massive data streams. J Comput Graph Stat 12:945–949

Giraud-Carrier C (2000) A note on the utility of incremental learning. AI Commun 13:215–223

Utgoff PE, Berkman NC, Clouse JA (1997) Decision tree induction based on efficient tree restructuring. Mach Learn 29:5–44

Indirect Reinforcement Learning

▶ Model-Based Reinforcement Learning

Induction

James Cussens
University of York, Heslington, UK

Definition

Induction is the process of inferring a general rule from a collection of observed instances. Sometimes it is used more generally to refer to any inference from premises to conclusion where the truth of the conclusion does not follow deductively from the premises, but where the premises provide evidence for the conclusion. In this more general sense, induction includes *abduction* where facts rather than rules are inferred. (The word "induction" also denotes a different, entirely deductive form of argument used in mathematics.)

Theory

Hume's Problem of Induction

The *problem of induction* was famously set out by the great Scottish empiricist philosopher David Hume (1711–1776), although he did not actually use the word "induction" in this context. With characteristic bluntness, he argued that:

> there can be no *demonstrative* arguments to prove *that those instances of which we have had no experience resemble those of which we have had experience* (Hume 1739, Part 3, Section 6).

Since scientists (and machine-learning algorithms) *do* infer future-predicting general laws from past observations, Hume is led to the following unsettling conclusion concerning human psychology (and statistical inference):

> It is not, therefore, reason, which is the guide of life, but custom. That alone determines the mind, in all instances, to suppose the future conformable to the past (Hume 1740).

That general laws cannot be demonstrated (i.e., deduced) from data is generally accepted. Hume, however, goes further: he argues that past observations do not even affect the *probability* of future events:

> Nay, I will go farther, and assert, that he could not so much as prove by any *probable* arguments, that the future must be conformable to the past. All probable arguments are built on the supposition, that there is this conformity betwixt the future and the past, and therefore can never prove it. This conformity is a *matter of fact*, and if it must be proved, will admit of no proof but from experience. But our experience in the past can be a proof of nothing for the future, but upon a supposition, that there is a resemblance betwixt them. This therefore is a point, which can admit of no proof at all, and which we take for granted without any proof (Hume 1740).

Induction and Probabilistic Inference

Hume's unwavering skepticism concerning prediction appears at variance with the predictive accuracy of machine learning algorithms: there is much experimental evidence that ML algorithms, once trained on "past observations," make predictions on unseen cases with an accuracy far in excess of what can be expected by chance. This apparent discrepancy between Hume's philosophy and practical experience of statistical inference can be explored using a familiar example from the literature on induction. Let e be the statement that all swans seen so far have been white and let h be the general rule that all swans are white. Since h implies e it follows that $P(e|h) = 1$ and so, using Bayes' theorem, we have that

$$P(h|e) = \frac{P(h)P(e|h)}{P(e)} = \frac{P(h)}{P(e)}. \qquad (1)$$

So $P(h|e) > P(h)$ as long as $P(e) < 1$ and $P(h) > 0$. This provides an explanation for the predictive accuracy of hypotheses supported by data: given supporting data they just have increased probability of being true. Of course, most machine learning outputs are not "noise-free" rules like h; almost always hypotheses claim a certain distribution for future data where no particular observation is ruled out entirely – some are just more likely than others. The same basic argument applies: if $P(h) > 0$ then as long as the observed data is more likely given the hypothesis than it is a priori, that is, as long as $P(e|h)/P(e) > 1$, then the probability of h will increase. Even in the (common) case where each hypothesis in the hypothesis space depends on real-valued parameters and so $P(h) = 0$ for all h, Bayes theorem still produces an increase in the probability *density* in the neighborhoods of hypotheses supported by the data.

In all these cases, it appears that e is giving "inductive support" to h. Consider, however, $h\prime$ which states that all swans until now have been white and *all future swans will be black*. Even in this case, we have that $P(h'|e) > P(h')$ as long as $P(e) < 1$ and $P(h') > 0$, though h and h' make entirely contradictory future predictions. This is a case of Goodman's paradox. The paradox is the result of confusing probabilistic inference with inductive inference. Probabilistic inference, of which Bayes theorem is an instance, is entirely deductive in nature – the conclusions of all probabilistic inferences follow with absolute certainty from their premises (and the axioms of probability). $P(h|e) > P(h)$ for $P(e) < 1$ and $P(h) > 0$ essentially because e has (deductively) ruled out some data that might have refuted h, not because a "conformity betwixt the future and the past" has been established.

Good performance on unseen data can still be explained. Statistical models (equivalently machine learning algorithms) *make assumptions* about the world. These assumptions (so far!) often turn out to be correct. Hume noted that the principle "that like objects, placed in like circumstances, will always produce like effects" (Hume 1739, Part 3, Section 8) although not deducible from first principles, has been established by "sufficient custom." This is called the *uniformity of nature* principle in the philosophical literature. It is this principle which informs machine learning systems. Consider the standard problem of predicting class labels for attribute-value data using labeled data as training. If an unlabeled test case has attribute values which are "close" to those of many training examples all of which have the same class label then in most systems the test case will be labeled also with this class. Different systems differ in how they measure "likeness": they differ in their ▶ inductive bias. A system which posited h' above on the basis of e would have an inductive bias strongly at variance with the uniformity of nature principle.

These issues resurfaced within the machine learning community in the 1990s. This ML work focused on various "▶ *no-free-lunch theorems*." Such a theorem essentially states that a uniformity of nature assumption is required to justify any given inductive bias. This is how Wolpert puts in one of the earliest "no-free-lunch" papers:

> This paper proves that it is impossible to justify a correlation between reproduction of a training set and generalization error off of the training set using only a priori reasoning. As a result, the use in the real world of any generalizer which fits a hypothesis function to a training set (e.g., the use of back-propagation) is implicitly predicated on an assumption about the physical universe (Wolpert 1992).

Note that in Bayesian approaches inductive bias is encapsulated in the prior distribution: once a prior has been determined all further work in Bayesian statistics is entirely deductive. Therefore it is no surprise that inductivists have sought to find "objective" or "logical" prior distributions to provide a firm basis for inductive inference. Foremost among these is Rudolf Carnap (1891–1970) who followed a logical approach – defining prior distributions over "possible worlds" (first-order models) which were in some sense uniform (Carnap 1950). A modern extension of this line of thinking can be found in Bacchus et al. (1996).

Popper

Karl Popper (1902–1994) accepted the Humean position on induction yet sought to defend science from charges of irrationality (Popper 1934). Popper *replaced* the problem of induction by the problem of criticism. For Popper, scientific progress proceeds by conjecturing universal laws and then subjecting these laws to severe tests with a view to refuting them. According to the *verifiability principle* of the logical positivist tradition, a theory is scientific if it can be experimentally confirmed, but for Popper confirmation is a hopeless task, instead a hypothesis is only scientific if it is *falsifiable*. All universal laws have prior probability of zero, and thus will eternally have probability zero of being true, no matter how many tests they pass. The value of a law can only be measured by how well-tested it is. The degree to which a law has been tested is called its degree of *corroboration* by Popper. The $P(e|h)/P(e)$ term in Bayes theorem will be high if a hypothesis h has passed many severe tests.

Popper's critique of inductivism continued throughout his life. In the *Popper–Miller* argument (Popper and Miller 1984), as it became known, it is observed that a hypothesis h is logically equivalent to:

$$(h \leftarrow e) \wedge (h \vee e)$$

for any evidence e. We have that $e \vdash h \vee e$ (where $\vdash$ means "logically implies") and also that (under weak conditions) $p(h \leftarrow e|e) < p(h \leftarrow e)$. From this Popper and Miller argue that

> ... we find that what is left of h once we discard from it everything that is logically implied by e, is a proposition that in general is counterdependent on e (Popper and Miller 1987)

and so.

> Although evidence may raise the probability of a hypothesis above the value it achieves on background knowledge alone, every such increase in probability has to be attributed entirely to the *deductive connections* that exist between the hypothesis and the evidence (Popper and Miller 1987).

In other words if $P(h|e) > P(h)$ this is only because $e \vdash h \vee e$. The Popper–Miller argument found both critics and supporters. Two basic arguments of the critics were that (1) deductive relations only set limits to probabilistic support; infinitely many probability distributions can still be defined on any given fixed system of propositions and (2) Popper–Miller are mischaracterizing induction as the absence of deductive relations, when it actually means *ampliative inference*: concluding more than the premises entail (Cussens 1996).

Causality and Hempel's Paradox

The branch of philosophy concerned with how evidence can confirm scientific hypotheses is known as ▸ *confirmation theory*. Inductivists take the position (against Popper) that observing data which follows from a hypothesis not only fails to refute the hypothesis, but also *confirms* it to some degree: seeing a white swan confirms the hypothesis that all swans are white, because

$$\begin{aligned}\forall x:\ \mathrm{swan}(x) \to \mathrm{white}(x), \mathrm{swan}(\mathrm{white\_swan})\\ \vdash \mathrm{swan}(\mathrm{white\_swan}).\end{aligned}$$

But, by the same argument it follows that observing any nonwhite, nonswan (say a black raven) also confirms that all swans are white, since:

$$\begin{aligned}\forall x:\ \mathrm{swan}(x) \to \mathrm{white}(x), \neg\mathrm{white}(\mathrm{black\_reven})\\ \vdash \neg(\mathrm{black\_reven}).\end{aligned}$$

This is Hempel's paradox to which there are a number of possible responses. One option is to accept that the black raven is a confirming instance, as one object in the universe has been ruled out as a potential refuter. The *degree* of confirmation is however of "a miniscule and negligible degree" (Howson and Urbach 1989, p. 90). Another option is to reject the formulation of the hypothesis as a material implication where $\forall x:\ \mathrm{swan}(x) \to \mathrm{white}(x)$ is just another way of writing $\forall x:\ \neg\mathrm{swan}(x) \vee \mathrm{white}(x)$. Instead, to be a scientific hypothesis of any interest the statement must be interpreted *causally*. This is the view of Imre Lakatos (1922–1974), and since any causal statement has a (perhaps implicit) *ceteris paribus* ("all other things being equal") clause this has implications for refutation also.

> . . . "all swans are white," if true, would be a mere curiosity unless it asserted that swanness *causes* whiteness. But then a black swan would not refute this proposition, since it may only indicate *other causes* operating simultaneously. Thus "all swans are white" is either an oddity and easily disprovable or a scientific proposition with a ceteris paribus clause and therefore easily undisprovable (Lakatos 1970, p. 102).

Cross-References

▸ Abduction
▸ Classification

Recommended Reading

Bacchus F, Grove A, Halpern JY, Koller D (1996) From statistical knowledge bases to degrees of belief. Artif Intell 87(1–2):75–143

Carnap R (1950) Logical foundations of probability. University of Chicago Press, Chicago

Cussens J (1996) Deduction, induction and probabilistic support. Synthese 108(1):1–10

Howson C, Urbach P (1989) Scientific reasoning: the Bayesian approach. Open Court, La Salle

Hume D (1739) A treatise of human nature, book one (Anonymously published)

Hume D (1740) An abstract of a treatise of human nature. (Anonymously published as a pamphlet). Printed for C. Borbet, London

Lakatos I (1970) Falsification and the methodology of scientific research programmes. In: Lakatos I, Musgrave A (eds) Criticism and the growth of knowledge. Cambridge University Press, Cambridge, pp 91–196

Popper KR (1959) The logic of scientific discovery. Hutchinson, London (Translation of *Logik der Forschung*, 1934)

Popper KR, Miller D (1984) The impossibility of inductive probability. Nature 310:434

Popper KR, Miller D (1987) Why probabilistic support is not inductive. Philos Trans R Soc Lond 321: 569–591

Wolpert DH (1992) On the connection between in-sample testing and generalization error. Complex Syst 6: 47–94

Induction as Inverted Deduction

▸ Logic of Generality

Inductive Bias

Synonyms

Learning bias; Variance hint

Definition

Most ML algorithms make predictions concerning future data which cannot be deduced from already observed data. The inductive bias of an algorithm is what choses between different possible future predictions. A strong form of inductive bias is the learner’s choice of hypothesis/model space which is sometimes called *declarative bias*. In the case of Bayesian analysis, the inductive bias is encapsulated in the prior distribution.

Cross-References

▶ Induction
▶ Learning as Search

Inductive Database Approach to Graphmining

Stefan Kramer
Technische Universität München, Garching b. München, Germany

Overview

The inductive database approach to graph mining can be characterized by (1) the concept of querying for (subgraph) patterns in databases of graphs, and (2) the use of specific data structures representing the space of solutions. For the former, a query language for the specification of the patterns of interest is necessary. The latter aims at a compact representation of the solution patterns.

Pattern Domain

In contrast to other graph mining approaches, the inductive database approach to graph mining (De Raedt and Kramer 2001; Kramer et al. 2001) focuses on simple patterns (paths and trees) and complex queries (see below), not on complex patterns (general subgraphs) and simple queries (minimum frequency only). While the first approaches were restricted to paths as patterns in graph databases, they were later extended toward unrooted trees (Rückert and Kramer 2003, 2004). Most of the applications are dealing with structures of small molecules and structure–activity relationships (SARs), that is, models predicting the biological activity of chemical compounds.

Query Language

The conditions on the patterns of interest are usually called *constraints* on the solution space. Simple constraints are specified by so-called *query primitives*. Query primitives express frequency-related or syntactic constraints. As an example, consider the frequency-related query primitive $f(p, D) \geq t$, meaning that a subgraph pattern p has to occur with a frequency of at least t in the database of graphs D. Analogously, other frequency-related primitives demand a maximum frequency of occurrence, or a minimum agreement with the target class (e.g., in terms of the information gain or the χ^2 statistic). Answering frequency-related queries generally requires database access. In contrast to frequency-related primitives, syntax-related primitives only restrict the syntax of solution (subgraph) patterns, and thus do not require database access. For instance, we may demand that a pattern p is more specific than “*c:c-Cl*” (formally $p \geq$ *c:c-Cl*) or more general than “*C-c:c:c:c:c-Cl*” (formally $p \leq$ *C-c:c:c:c:c-Cl*). The strings in the primitive contain vertex (e.g., “*C*,” “*c*,” “*Cl*”. . .) and edge labels (e.g., “ : ,” “-”. . .) of a path in a graph. Many constraints on patterns can be categorized as either monotonic or anti-monotonic. Minimum frequency constraints, for instance, are anti-monotonic, because all subpatterns (in our case: subgraphs) are frequent as well, if a pattern is frequent (according to some user-defined threshold) in a database. Vice versa, maximum frequency is monotonic, because if a pattern is not too frequent, then all superpatterns (in our case: supergraphs) are not too frequent either. Anti-monotonic or monotonic constraints can be solved by variants of levelwise search and APriori (De Raedt and Kramer 2001; Kramer et al. 2001; Mannila and Toivonen 1997). Other types of constraints involving convex functions, for example, related to the target class, can be solved by branch-and-bound algorithms (Morishita and Sese 2000). Typical query languages offer the possibility to combine query primitives conjunctively or disjunctively.

Data Structures

It is easy to show that solutions to conjunctions of monotonic and anti-monotonic constraints can be represented by *version spaces*, and in particular, borders of the most general and the most specific patterns satisfying the constraints (De Raedt and Kramer 2001; Mannila and Toivonen 1997). Version spaces of patterns can be represented in data structures such as *version space trees* (De Raedt et al. 2002; Rückert and Kramer 2003). For sequences, data structures based on *suffix arrays* are known to be more efficient than data structures based on version spaces (Fischer et al. 2006). Query languages allowing disjunctive normal forms of monotonic or anti-monotonic primitives yield multiple version spaces as solutions, represented by generalizations of version space trees (Lee and De Raedt 2003). The inductive database approach to graph mining can also be categorized as *constraint-based mining*, as the goal is to find solution patterns satisfying user-defined constraints.

Recommended Reading

De Raedt L, Kramer S (2001) The levelwise version space algorithm and its application to molecular fragment finding. In: Proceedings of the seventeenth international joint conference on artificial intelligence (IJCAI 2001). Morgan Kaufmann, San Francisco

De Raedt L, Jaeger M, Lee SD, Mannila H (2002) A theory of inductive query answering. In: Proceedings of the 2002 IEEE international conference on data mining (ICDM 2002). IEEE Computer Society, Washington, DC

Fischer J, Heun V, Kramer S (2006) Optimal string mining under frequency constraints. In: Proceedings of the tenth European conference on the principles and practice of knowledge discovery in databases (PKDD 2006). Springer, Berlin

Kramer S, De Raedt L, Helma C (2001) Molecular feature mining in HIV data. In: Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining (KDD 2001). ACM, New York

Lee SD, De Raedt L (2003) An algebra for inductive query evaluation. In: Proceedings of the third IEEE international conference on data mining (ICDM 2003). IEEE Computer Society, Washington, DC

Mannila H, Toivonen H (1997) Levelwise search and borders of theories in knowledge discovery. Data Min Knowl Discov 1(3):241–258

Morishita S, Sese J (2000) Traversing itemset lattice with statistical metric pruning. In: Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems (PODS 2000). ACM, New York

Rückert U, Kramer S (2003) Generalized version space trees. In: Boulicaut J-F, Dzeroski S (eds) Proceedings of the second international workshop on knowledge discovery in inductive databases (KDID-2003). Berlin, Springer

Rückert U, Kramer S (2004) Frequent free tree discovery in graph data. In: Proceedings of the ACM symposium on applied computing (SAC 2004). ACM, New York

Inductive Inference

Sanjay Jain[1] and Frank Stephan[2]
[1]School of Computing, National University of Singapore, Singapore, Singapore
[2]Department of Mathematics, National University of Singapore, Singapore, Singapore

Definition

Inductive inference is a theoretical framework to model learning in the limit. The typical scenario is that the learner reads successively datum

$d_0, d_1, d_2, \ldots$ about a concept and outputs in parallel hypotheses $e_0, e_1, e_2, \ldots$ such that each hypothesis e_n is based on the preceding data $d_0, d_1, \ldots, d_{n-1}$. The hypotheses are expected to converge to a description for the data observed; here the constraints on how the convergence has to happen depend on the learning paradigm considered. In the most basic case, almost all e_n have to be the same correct index e, which correctly explains the target concept. The learner might have some preknowledge of what the concept might be, that is, there is some class $\mathcal{C}$ of possible target concepts – the learner has only to find out which of the concepts in $\mathcal{C}$ is the target concept; on the other hand, the learner has to be able to learn every concept which is in the class $\mathcal{C}$.

Detail

The above given scenario of learning is essentially the paradigm of inductive inference introduced by Gold (1967) and known as **Ex** (explanatory) learning. Usually one considers learning of recursive functions or recursively enumerable languages. Intuitively, using coding, one can code any natural phenomenon into subsets of $\mathbb{N}$, the set of natural numbers. Thus, recursive functions from $\mathbb{N}$ to $\mathbb{N}$ or recursively enumerable subsets of $\mathbb{N}$ (called languages here) are natural concepts to be considered.

Here we will mainly consider language learning. Paradigms related to function learning can be similarly defined and we refer the reader to Osherson et al. (1986) and Jain et al. (1999).

One normally considers data provided to the learner to be either full positive data (i.e., the learner is told about every element in the target language, one element at a time, but never told anything about elements not in the target language) or full positive data and full negative data (i.e., the learner is told about every element, whether it belongs or does not belong to the target language). Intuitively, the reason for considering only positive data is that in many natural situations, such as language learning by children and scientific exploration (such as in astronomy), one gets essentially only positive data.

A *text* is a sequence of elements over $\mathbb{N} \cup \{\#\}$. Content of a text T, denoted $\mathrm{ctnt}(T)$, is the set of natural numbers in the range of T. For a finite sequence σ over $\mathbb{N} \cup \{\#\}$, one can similarly define $\mathrm{ctnt}(\sigma)$ as the set of natural numbers in the range of σ. A text T is said to be for a language L if $\mathrm{ctnt}(T) = L$. Intuitively, a text T for L represents sequential presentation of all elements of L, with #'s representing pauses in the presentation. For example, the only text for $\emptyset$ is $\#^\infty$. $T[n]$ denotes the initial sequence of T of length n. That is, $T[n] = T(0)T(1)\ldots T(n-1)$. We let SEQ denote the set of all finite sequences over $\mathbb{N} \cup \{\#\}$. An *informant* I is a sequence of elements over $\mathbb{N} \times \{0,1\} \cup \{\#\}$, where for each $x \in \mathbb{N}$, exactly one of $(x,0)$ or $(x,1)$ is in the range of I. An informant I is for L if $\mathrm{range}(I) - \{\#\} = \{(x, \chi_L(x)) : x \in \mathbb{N}\}$, where χ_L denotes the characteristic function of L.

A learner M is a mapping from SEQ to $\mathbb{N} \cup \{?\}$. Intuitively, output of ? denotes that the learner does not wish to make a conjecture on the corresponding input. The output of e denotes that the learner conjectures hypothesis W_e, where $W_0, W_1, \ldots$ is some acceptable numbering of all the recursively enumerable languages. We say that a learner M converges on T to e if, for all but finitely many n, $\mathrm{M}(T[n]) = e$.

Explanatory Learning

A learner M **TxtEx** identifies a language L iff, for all texts T for L, M converges to an index e such that $W_e = L$. Learner M **TxtEx** identifies a class $\mathcal{L}$ of languages if M **TxtEx** identifies each language in the class $\mathcal{L}$. Finally, one says that a class $\mathcal{L}$ is **TxtEx** learnable if some learner **TxtEx** identifies $\mathcal{L}$. **TxtEx** denotes the collection of all **TxtEx**-learnable classes. One can similarly define **InfEx** identification, for learning from informants instead of texts. The following classes are important examples:

$$RE = \{L : L \text{ is recursively enumerable}\};$$

$$FIN = \{L : L \text{ is a finite subset of } \mathbb{N}\};$$

$$KFIN = \{L : L = K \cup H \text{ for some } H \in FIN\};$$

$$SD = \{L : W_{\min(L)} = L\};$$

$$COFIN = \{L : \mathbb{N} - L \text{ is finite}\};$$

$$SDSIZE = \{\{e + x : x = 0 \vee x < |W_e|\} : W_e \text{ is finite}\};$$

$$SDALL = \{\{e + x : x \in \mathbb{N}\} : e \in \mathbb{N}\}.$$

Here, in the definition of *KFIN*, K is the halting problem, a standard example of a set which is recursively enumerable but not recursive. The classes *FIN*, *SD*, *SDSIZE*, and *SDALL* are **TxtEx** learnable (Case and Smith 1983; Gold 1967): The learner for *FIN* always conjectures the set of all data observed so far. The learner for *SD* conjectures the least datum seen so far as, eventually, the least observed datum coincides with the least member of the language to be learned. The learner for *SDSIZE* as well as the learner for *SDALL* also find in the limit the least datum e to occur and translate it into an index for the e-th set to be learned. The class *KFIN* is not **TxtEx** learnable, mainly for computational reasons. It is impossible for the learner to determine if the current input datum belongs to K or not; this forces a supposed learner either to make infinitely many mind changes on some text for K or to make an error on $K \cup \{x\}$, for some $x \notin K$. The union *SDSIZE* ∪ *SDALL* is also not **TxtEx** learnable, although it is the union of two learnable classes; so it is one example of various nonunion theorems. Gold (1967) gave even a more basic example: *FIN* ∪ $\{\mathbb{N}\}$ is not **TxtEx** learnable. Furthermore, the class *COFIN* is also not **TxtEx** learnable. However, except for *RE*, all the classes given above are **InfEx** learnable, so when being fed the characteristic function in place of only an infinite list of all elements, the learners become, in general, more powerful.

Note that the learner never knows when it has converged to its final hypothesis. If the learner is required to know when it has converged to the final hypothesis, then the criterion of learning is the same as finite learning. Here a finite learner is defined as follows: the learner keeps outputting the symbol ? while waiting for enough data to appear and, when the data observed are sufficient, the learner outputs exactly one conjecture different from ?, which then is required to be an index for the input concept in the hypothesis space. The class of singletons $\{\{n\} : n \in \mathbb{N}\}$ is finitely learnable; the learner just waits until the unique element n of $\{n\}$ has appeared and then knows the language. In contrast to this, the classes *FIN* and *SD* are not finitely learnable.

Blum and Blum (1975) obtained the following fundamental result: Whenever M learns L explanatorily from text, then L has a locking sequence for M. Here, a sequence σ is said to be a locking sequence for M on L if (a) $\text{ctnt}(\sigma) \subseteq L$, (b) for all τ such that $\text{ctnt}(\tau) \subseteq L$, $\text{M}(\sigma) = \text{M}(\sigma\tau)$, and (c) $W_{\text{M}(\sigma)} = L$. If only the first two conditions are satisfied, then the sequence is called a *stabilizing sequence* for M on L (Fulk 1990). It was shown by Blum and Blum (1975) that if a learner M **TxtEx** identifies L, then there exists a locking sequence σ for M on L. One can use this result to show that certain classes, such as *FIN* ∪ $\{\mathbb{N}\}$, are not **TxtEx** learnable.

Beyond Explanatory Learning

While **TxtEx** learning requires that the learner syntactically converges to a final hypothesis, which correctly explains the concept, this is no longer required for the more general criterion of behaviorally correct learning (called **TxtBc** learning). Here, the learner may not syntactically converge, but it is still required that all its hypothesis after sometime are correct; see Bārzdiņš (1974b), Case and Lynes (1982), Case and Smith (1983), Osherson et al. (1986), and Osherson and Weinstein (1982). So there is semantic convergence to a final hypothesis. Thus, a learner M **TxtBc** identifies a language L if for all texts T for L, for all but finitely many n, $W_{\text{M}(T[n])} = L$. One can similarly define **TxtBc** learnability of classes of languages and the collection **TxtBc**. Every **TxtEx**-learnable class is **Bc** learnable, but the classes *KFIN* and *SDSIZE* ∪ *SDALL* are **TxtBc** learnable but not **TxtEx** learnable. Furthermore, **InfEx** $\not\subseteq$ **TxtBc**, for example, *FIN* ∪ $\{\mathbb{N}\}$ is **InfEx** learnable but not **TxtBc** learnable. On the other hand, every class that is finitely learnable from informant is also **TxtEx** learnable (Sharma 1998).

An intermediate learning criterion is **TxtFex** learning (Case 1999) or vacillatory learning, which is similar to **TxtBc** learning except that we require that the number of distinct hypotheses output by the learner on any text is finite. Here one says that the learner $\mathbf{TxtFex}_n$ learns the language L if the number of distinct hypotheses that appears infinitely often on any text T for L is bounded by n. Note that $\mathbf{TxtFex}_* = \mathbf{TxtFex}$. Case (1999) showed that

$$\mathbf{TxtEx} = \mathbf{TxtFex}_1 \subset \mathbf{TxtFex}_2 \subset \mathbf{TxtFex}_3 \subset \ldots \subset \mathbf{TxtFex}_* \subset \mathbf{TxtBc}.$$

For example, the class $SD \cup SDALL$ is actually $\mathbf{TxtFex}_2$ learnable and not **TxtEx** learnable. The corresponding notion has also been considered for function learning, but there the paradigms of explanatory and vacillatory learning coincide (Case and Smith 1983).

Blum and Blum (1975), Case and Lynes (1982), and Case and Smith (1983) also considered allowing the final (or final sequence of) hypothesis to be anomalous; Blum and Blum (1975) considered $*$ anomalies, and (Case and Lynes 1982; Case and Smith 1983) considered the general case. Here the final grammar for the input language may not be perfect, but may have up to a anomalies. A grammar n is a anomalous for L (written $W_n =^a L$) iff card $((L - W_n) \cup (W_n - L)) \leq a$. Here one also considers finite anomalies, denoted by $*$-anomalies, where card$(S) \leq *$ just means that S is finite. Thus, a learner M $\mathbf{TxtEx}^a$ identifies a language L iff, for all texts T for all L, M on T converges to a hypothesis e such that $W_e =^a L$. One can similarly define $\mathbf{TxtBc}^a$-learning criteria. It can be shown that

$$\mathbf{TxtEx} = \mathbf{TxtEx}^0 \subset \mathbf{TxtEx}^1 \subset \mathbf{TxtEx}^2 \subset \ldots \subset \mathbf{TxtEx}^*$$

and

$$\mathbf{TxtBc} = \mathbf{TxtBc}^0 \subset \mathbf{TxtBc}^1 \subset \mathbf{TxtBc}^2 \subset \ldots \subset \mathbf{TxtBc}^*.$$

Let $SD_n = \{L : W_{\min(L)} =^n L\}$. Then one can show (Case and Lynes 1982; Case and Smith 1983) that $SD_{n+1} \in \mathbf{TxtEx}^{n+1} - \mathbf{TxtEx}^n$. However, there is a trade-off between behaviorally correct learning and explanatory learning for learning with anomalies. On one hand, $\mathbf{TxtBc} \not\subseteq \mathbf{TxtEx}^*$, but on the other hand $\mathbf{TxtEx}^{2n+1} \not\subseteq \mathbf{TxtBc}^n$ and $\mathbf{TxtEx}^{2n} \subseteq \mathbf{TxtBc}^n$. However, for learning from informants, we have $\mathbf{InfEx}^* \subseteq \mathbf{InfBc}$ (see Case and Lynes (1982) for the above results).

Consistent and Conservative Learning

Besides the above basic criteria of learning, researchers have also considered several properties that are useful for the learner to satisfy.

A learner M is said to be *consistent* on L iff, for all texts T for L, ctnt$(T[n]) \subseteq W_{\mathrm{M}(T[n])}$. That is, the learner's hypothesis is consistent with the data seen so far. There are three notions of consistency considered in the literature: (a) **TCons**, in which the learner is expected to be consistent on all inputs, irrespective of whether they represent some concept from the target class or not (Wiehagen and Liepe 1976); (b) **Cons**, in which the learner is just expected to be consistent on the languages in the target class being learned, though the learner may be inconsistent or even undefined on the input outside the target class (Bārzdiņš 1974a); and (c) **RCons**, in which the learner is expected to be defined on all inputs, but required to be consistent only on the languages in the target class (Jantke and Beick 1981). It can be shown that $\mathbf{TCons} \subset \mathbf{RCons} \subset \mathbf{Cons} \subset \mathbf{TxtEx}$ (Jantke and Beick 1981; Wiehagen and Liepe 1976; Bārzdiņš 1974a; Wiehagen and Zeugmann 1995).

A learner M is said to be *conservative* (Angluin 1980) if it does not change its mind unless the data contradicts its hypothesis. That is, M conservatively learns L iff, for all texts T for L, if $\mathrm{M}(T[n]) \neq \mathrm{M}(T[n+1])$, then ctnt$(T[n+1]) \not\subseteq W_{\mathrm{M}(T[n])}$. It can be shown that conservativeness is restrictive, that is, there are classes of languages, which can be **TxtEx** identified but not conservatively identified. An example of a class that can be identified explanatorily but not conservatively is the class containing all sets from *SDALL*, that is, the sets of the form $\{e, e+1, e+2, \ldots\}$, and all sets with minimum k_s and up to s elements where $k_0, k_1, k_2, \ldots$ is a recursive one-one enumeration of K. The general idea why this class is not conservatively learnable is that when the learner reads the data $e, e+1, e+2, \ldots$, it will, after some finite time based on data $e, e+1, e+2, \ldots, e+s$, output a conjecture which contains these data plus $e+s+1$; but conservative learning would then imply that $e \in K$ iff $e = k_r$ for some $r \leq s$, contradicting the non-recursiveness of K.

Monotonicity

Related notions to conservativeness are the various notions on monotonic learning that impose certain conditions on whether the previous hypothesis is a subset of the next hypothesis or not. The following notions are the three main ones.

- A learner M is said to be strongly monotonic (Jantke 1991) on L iff, for all texts T for L, $W_{\mathrm{M}(T[n])} \subseteq W_{\mathrm{M}(T[n+1]}$. Intuitively, strong monotonicity requires that the hypothesis of the learner grows with time.
- A learner M is said to be monotonic (Wiehagen 1990) on L iff, for all texts T for L, $W_{\mathrm{M}(T[n])} \cap L \subseteq W_{\mathrm{M}(T[n+1])} \cap L$. In monotonicity, the growth of the hypothesis is required only with respect to the language being learned.
- A learner M is said to be weakly monotonic (Jantke 1991) on L iff, for all texts T for L, if $\mathrm{ctnt}(T[n+1]) \subseteq W_{\mathrm{M}(T[n])}$, then $W_{\mathrm{M}(T[n])} \subseteq W_{\mathrm{M}(T[n+1])}$. That is, the learner behaves strongly monotonically, as long as the input data is consistent with the hypothesis.

An example for a strong monotonically learnable class is the class *SDALL*. When the learner currently conjectures $\{e, e+1, e+2, \ldots\}$ and it sees a datum $d < e$, then it makes a mind change to $\{d, d+1, d+2, \ldots\}$ which is a superset of the previous conjecture; it is easy to see that all mind changes are of this type. It can be shown that strong monotonic learning implies monotonic learning and weak monotonic learning, though monotonic learning and weak monotonic learning are incomparable (and thus both are proper restrictions of **TxtEx** learning). For example, consider the class $\mathcal{C}$ consisting of the set $\{0, 2, 4, \ldots\}$ of all even numbers and, for each n, the set $\{0, 2, 4, \ldots, 2n\} \cup \{2n+1\}$ consisting of the even numbers below $2n$ and the odd number $2n+1$. Then, $\mathcal{C}$ is monotonically but not strong monotonically learnable.

Lange et al. (1992) also considered the dual version of the above criteria, where dual strong monotonicity learning of L requires that, for all texts T for L, $W_{\mathrm{M}(T[n])} \supseteq W_{\mathrm{M}(T[n+1])}$; dual monotonicity requires that, for all texts T for L, $W_{\mathrm{M}(T[n])} \cap (\mathbb{N} - L) \supseteq W_{\mathrm{M}(T[n+1])} \cap (\mathbb{N} - L)$, and dual weak monotonicity requires that, if $\mathrm{ctnt}(T[n+s]) \subseteq W_{\mathrm{M}(T[n])}$, then $W_{\mathrm{M}(T[n])} \supseteq W_{\mathrm{M}(T[n+s])}$.

In a similar fashion, various other properties of learners have been considered. For example, reliability (Blum and Blum 1975; Minicozzi 1976) postulates that the learner does not converge on the input text unless it learns it; prudence (Fulk 1990; Osherson et al. 1986) postulates that the learner outputs only indices of languages, which it also learns; and confidence (Osherson et al. 1986) postulates that the learner converges on every text to some index, even if the text is for some language outside the class of languages to be learned.

Indexed Families

Angluin (1980) initiated a study of learning indexed families of recursive languages. A class of languages (along with its indexing) $L_0, L_1, \ldots$ is an indexed family if membership questions for the languages are uniformly decidable, that is, $x \in L_i$ can be recursively decided in x and i. Angluin gave an important characterization of indexed families that are **TxtEx** learnable.

Suppose a class $\mathcal{L} = \{L_0, L_1, \ldots\}$ (along with the indexing) is given. Then, S is said to be a *tell-tale set* (Angluin 1980) of L_i iff S is finite, and for all j, if $S \subseteq L_j$ and $L_j \subseteq L_i$, then $L_i = L_j$. It can be shown that for any class of languages that are learnable (in **TxtEx** or **TxtBc** sense), there exists a tell-tale set for each language in the class. Moreover, Angluin showed that for indexed families, $\mathcal{L} = L_0, L_1, \ldots$, one can **TxtEx** learn $\mathcal{L}$ iff one can recursively enumerate a tell-tale set for each L_i, effectively from i. Within the framework of learning indexed families, a special emphasis is given to the hypothesis space used; so the following criteria are considered for defining the learnability of a class $\mathcal{L}$ in dependence of the hypothesis space $\mathcal{H} = H_0, H_1, \ldots$. The class $\mathcal{L}$ is

- *Exactly learnable* iff there is a learner using the same hypothesis space as the given class, that is, $H_n = L_n$ for all n;
- *Class-preservingly learnable* iff there is a learner using a hypothesis space $\mathcal{H}$ with $\{L_0, L_1, \ldots\} = \{H_0, H_1, \ldots\}$ – here the order and the number of occurrences in the hypothesis space can differ, but the hypothesis space must consist of the same languages as the class to be learned, and no other languages are allowed in the hypothesis space;
- *Class-comprisingly learnable* iff there is a learner using a hypothesis space $\mathcal{H}$ with $\{L_0, L_1, \ldots\} \subseteq \{H_0, H_1, \ldots\}$ – here the hypothesis space can also contain some further languages not in the class to be learned and the learner does not need to identify these additional languages;
- *Prescribed learnable* iff for every hypothesis space $\mathcal{H}$ containing all the languages from $\mathcal{L}$, there is a learner for $\mathcal{L}$ using this hypothesis space;
- *Uniformly learnable* iff for every hypothesis space $\mathcal{H}$ with index e containing all the languages from $\mathcal{L}$ one can synthesize a learner M_e which succeeds to learn $\mathcal{L}$ using the hypothesis space $\mathcal{H}$.

Note that in all five cases $\mathcal{H}$ only ranges over indexed families. This differs from the standard case where $\mathcal{H}$ is an acceptable numbering of all recursively enumerable sets. We refer the reader to the survey of Lange et al. (2008) for an overview on work done on learning indexed families (**TxtEx** learning, learning under various properties of learners, as well as characterizations of such learning criteria) and to (Jain et al. 2008; Lange and Zeugmann 1993). While for explanatory learning and every class $\mathcal{L}$, all these five notions coincide, these notions turn out to be different for other learning notions like those of conservative learning, monotonic learning, and strong monotonic learning. For example, the class of all finite sets is not prescribed conservatively learnable: one can make an adversary hypothesis space where some indices contain large spurious elements, so that a learner is forced to do nonconservative mind change to obtain correct indices for the finite sets. The same example as above works for showing the limitations of prescribed learning for monotonic and strong monotonic learning.

The interested reader is referred to the textbook *Systems that Learn* (Jain et al. 1999; Osherson et al. 1986) and the papers below as well as the references found in these papers for further reading. Complexity issues in inductive inference like the number of mind changes necessary to learn a class or oracles needed to learn some class can be found under the entries *Computational Complexity of Learning* and *Query-Based Learning*. The entry *Connections between Inductive Inference and Machine Learning* provides further information on this topic.

Cross-References

▸ Connections Between Inductive Inference and Machine Learning

Acknowledgements Sanjay Jain was supported in part by NUS grant numbers C252-000-087-001, R146-000-181-112, R252-000-534-112. Frank Stephen was supported in part by NUS grant numbers R146-000-181-112, R252-000-534-112.

Recommended Reading

Angluin D (1980) Inductive inference of formal languages from positive data. Inf Control 45:117–135

Bārzdiņš J (1974a) Inductive inference of automata, functions and programs. In: Proceedings of the international congress of mathematics, Vancouver, pp 771–776

Bārzdiņš J (1974b) Two theorems on the limiting synthesis of functions. In: Theory of algorithms and programs, vol 1. Latvian State University, Riga, pp 82–88 (In Russian)

Blum L, Blum M (1975) Toward a mathematical theory of inductive inference. Inf Control 28:125–155

Case J (1999) The power of vacillation in language learning. SIAM J Comput 28:1941–1969

Case J, Lynes C (1982) Machine inductive inference and language identification. In: Nielsen M, Schmidt EM (eds) Proceedings of the 9th international colloquium on automata, languages and programming. Lecture notes in computer science, vol 140. Springer, Heidelberg, pp 107–115

Case J, Smith C (1983) Comparison of identification criteria for machine inductive inference. Theor Comput Sci 25:193–220

Fulk M (1990) Prudence and other conditions on formal language learning. Inf Comput 85:1–11

Gold EM (1967) Language identification in the limit. Inf Control 10:447–474

Jain S, Osherson D, Royer J, Sharma A (1999) Systems that learn: an introduction to learning theory, 2nd edn. MIT Press, Cambridge

Jain S, Stephan F, Ye N (2008) Prescribed learning of indexed families. Fundam Inf 83:159–175

Jantke KP (1991) Monotonic and non-monotonic inductive inference. New Gener Comput 8:349–360

Jantke KP, Beick H-R (1981) Combining postulates of naturalness in inductive inference. J Inf Process Cybern (EIK) 17:465–484

Lange S, Zeugmann T (1993) Language learning in dependence on the space of hypotheses. In: Proceedings of the sixth annual conference on computational learning theory, Santa Cruz, pp 127–136

Lange S, Zeugmann T, Kapur S (1992) Class preserving monotonic language learning. Technical report 14/92, GOSLER-Report, FB Mathematik und Informatik, TH Leipzig

Lange S, Zeugmann T, Zilles S (2008). Learning indexed families of recursive languages from positive data: a survey. Theor Comput Sci 397: 194–232

Minicozzi E (1976) Some natural properties of strong identification in inductive inference. Theor Comput Sci 2:345–360

Osherson D, Weinstein S (1982) Criteria of language learning. Inf Control 52:123–138

Osherson D, Stob M, Weinstein S (1986) Systems that learn, an introduction to learning theory for cognitive and computer scientists. Bradford–The MIT Press, Cambridge

Sharma A (1998) A note on batch and incremental learnability. J Comput Syst Sci 56:272–276

Wiehagen R (1990) A thesis in inductive inference. In: Dix J, Jantke K, Schmitt P (eds) Nonmonotonic and inductive logic, 1st international workshop. Lecture notes in artificial intelligence, vol 543. Springer, Berlin, pp 184–207

Wiehagen R, Liepe W (1976) Charakteristische Eigenschaften von erkennbaren Klassen rekursiver Funktionen. J Inf Process Cybern (EIK) 12:421–438

Wiehagen R, Zeugmann T (1995) Learning and consistency. In: Jantke KP, Lange S (eds) Algorithmic learning for knowledge-based systems (GOSLER), final report. Lecture notes in artificial intelligence, vol 961. Springer, Heidelberg, pp 1–24

Inductive Inference Rules

▶ Logic of Generality

Inductive Learning

Synonyms

Statistical learning

Definition

Inductive learning is a subclass of machine learning that studies algorithms for learning knowledge based on statistical regularities. The learned knowledge typically has no deductive guarantees of correctness, though there may be statistical forms of guarantees.

Inductive Logic Programming

Luc De Raedt
Department of Computer Science, Katholieke
Universiteit Leuven, Heverlee, Leuven, Belgium

Abstract

Inductive logic programming is the subfield of machine learning that uses ▶ First-Order Logic to represent hypotheses and data.

Because first-order logic is expressive and declarative, inductive logic programming specifically targets problems involving structured data and background knowledge. Inductive logic programming tackles a wide variety of problems in machine learning, including classification, regression, clustering, and reinforcement learning, often using "upgrades" of existing propositional machine learning systems. It relies on logic for knowledge representation and reasoning purposes. Notions of coverage, generality, and operators for traversing the space of hypotheses are grounded in logic; see also ▸ Logic of Generality. Inductive logic programming systems have been applied to important applications in bio- and chemoinformatics, natural language processing, and web mining.

Synonyms

Learning in logic; Multi-relational data mining; Relational data mining; Relational learning

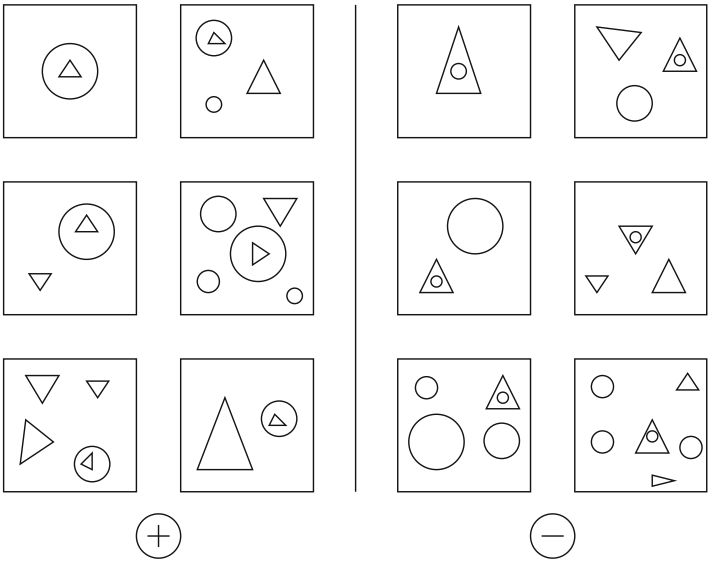
Motivation

The first motivation and most important motivation for using inductive logic programming is that it overcomes the representational limitations of attribute-value learning systems. Such systems employ a table-based representations where the instances correspond to rows in the table, the attributes to columns, and for each instance, a single value is assigned to each of the attributes. This is sometimes called the *single-table single-tuple* assumption. Many problems, such as the Bongard problem shown in Fig. 1, cannot elegantly be described in this format. Bongard (1970) introduced about a hundred concept learning or pattern recognition problems, each containing six positive and six negative examples. Even though Bongard problems are toy problems, they are similar to real-life problems such as structure–activity relationship prediction, where the goal is to learn to predict whether a given molecule (as represented by its 2D graph structure) is active or not. It is hard – if not, impossible – to squeeze this type of problem into the single-table single-tuple format for various reasons. Attribute-value learning systems employ a fixed number of attributes and also assume that these attributes are present in all of the examples. This assumption does not hold for the Bongard problems as the examples possess a variable number of objects (shapes). The singe-table single-tuple representation imposes an implicit order on the attributes, whereas there is no natural order on the objects in the Bongard problem. Finally, the relationships between the objects in the Bongard problem are essential and must be encoded as well. It is unclear how to do this within the single-table single-tuple assumption. First-order logic and relational representations allow one to encode problems involving multiple objects (or entities) as well as the relationships that hold them in a natural way.

The second motivation for using inductive logic programming is that it employs logic, a declarative representation. This implies that hypotheses are understandable and interpretable. By using logic, inductive logic programming systems are also able to employ background knowledge in the induction process. Background knowledge can be provided in the form of definitions of auxiliary relations or predicates that may be used by the learner. Finally, logic provides a well-understood theoretical framework for knowledge representation and reasoning. This framework is also useful for machine learning, in particular, for defining and developing notions such as the covers relation, generality, and refinement operators; see also ▸ Logic of Generality.

Theory

Inductive logic programming is usually defined as concept learning using logical representations. It aims at finding a hypothesis (a set of rules) that covers all positive examples and none of the negatives, while taking into account a background theory. This is typically realized by searching a



Inductive Logic Programming, Fig. 1 A complex classification problem: Bongard problem 47, developed by the Russian scientist Bongard (1970). It consists of 12 scenes (or examples), 6 of class ⊕ and 6 of class ⊖. The goal is to discriminate between the two classes

space of possible hypotheses. More formally, the traditional inductive logic programming definition reads as follows:

Given

- A language describing hypotheses $\mathcal{L}_h$
- A language describing instances $\mathcal{L}_i$
- Possibly a background theory B, usually in the form of a set of (definite) clauses
- The *covers* relation that specifies the relation between $\mathcal{L}_h$ and $\mathcal{L}_i$, that is, when an example e is covered (considered positive) by a hypothesis h, possibly taking into account the background theory B
- A set of positive and negative examples $E = P \cup N$

Find a hypothesis $h \in \mathcal{L}_h$ such that for all $p \in P : covers(B, h, p) = true$ and for all $n \in N : covers(B, h, n) = false$.

This definition can, as for ▶ Concept-Learning in general, be extended to cope with noisy data by relaxing the requirement that all examples be classified correctly.

There exist different ways to represent learning problems in logic, resulting in different learning settings. They typically use definite clause logic as the hypothesis language $\mathcal{L}_i$ but differ in the notion of an example. One can learn from entailment, from interpretations, or from proofs, cf. ▶ Logic of Generality. The most popular setting is *learning from entailment*, where each example is a clause and $covers(B, h, e) = true$ if and only if $B \cup h \models e$.

The top leftmost scene in the Bongard problem of Fig. 1 can be represented by the clause:

```
positive :- object(o1),
            object(o2),
            circle(o1),
            triangle(o2),
            in(o1, o2),
            large(o2).
```

The other scenes can be encoded in the same way. The following hypothesis then forms a solution to the learning problem:

```
positive :- object(X),
            object(Y),
            circle(X),
            triangle(Y),
            in(X,Y).
```

It states that those scenes having a circle inside a triangle are positive. For some more complex Bongard problems, it could be useful to employ background knowledge. It could, for instance, state that triangles are polygons.

```
polygon(X) :- triangle(X).
```

Using this clause as background theory, an alternative hypothesis covering all positives and none of the negatives is

```
positive :- object(X),
            object(Y),
            circle(X),
            polygon(Y),
            in(X,Y).
```

An alternative for using long clauses as examples is to provide an identifier for each example and to add the corresponding facts from the condition part of the clause to the background theory. For the above example, the facts such as

```
object(e1,o1).
object(e1,o2).
circle(e1,o1).
triangle(e1,o2).
in(e1,o1,o2).
large(e1,o2).
```

would be added to the background theory, and the positive example itself would then be represented through the fact `positive(e1)`, where `e1` is the identifier. The inductive logic programming literature typically employs this format for examples and hypotheses.

Whereas inductive logic programming originally focused on concept learning – as did the whole field of machine learning – it is now being applied to virtually all types of machine learning problems, including regression, clustering, distance-based learning, frequent pattern mining, reinforcement learning, and even kernel methods and graphical models.

A Methodology

Many of the more recently developed inductive logic programming systems have started from an existing attribute-value learner and have upgraded it toward the use of first-order logic (Van Laer and De Raedt 2001). By examining state-of-the-art inductive logic programming systems, one can identify a methodology for realizing this (Van Laer and De Raedt 2001). It starts from an attribute-value learning problem and system of interest and takes the following two steps. First, the problem setting is upgraded by changing the representation of the examples, the hypotheses as well as the covers relation toward first-order logic. This step is essentially concerned with defining the learning setting, and possible settings to be considered include the already mentioned learning from *entailment*, *interpretations*, and *proofs* settings. Once the problem is clearly defined, one can attempt to formulate a solution. Thus, the second step adapts the original algorithm to deal with the upgraded representations. While doing so, it is advisable to keep the changes as minimal as possible. This step often involves the modification of the operators used to traverse the search space. Different operators for realizing this are introduced in the entry on the ▶ Logic of Generality.

There are many reasons why following the methodology is advantageous. First, by upgrading a learner that is already effective for attribute-value representations, one can benefit from the experiences and results obtained in the propositional setting. In many cases, for instance, decision trees, this implies that one can rely on well-established methods and findings, which are the outcomes of several decades of machine learning research. It will be hard to do better starting from scratch. Second, upgrading an existing learner is also easier than starting from scratch as many of the components (such as heuristics and search strategy) can be recycled. It is therefore also economic in terms of man power. Third, the upgraded system will be able to emulate the original one, which provides guarantees that the output hypotheses will perform well on attribute-value learning problems. Even more important is that it will often also be able to emulate extensions of the original systems. For instance, many systems that extend frequent item-set mining toward using richer representations, such as sequences, intervals, the use of taxonomies, graphs, and so on, have been developed over the past decade. Many of them can be emulated using the inductive logic programming upgrade of Apriori (Agrawal et al. 1996) called Warmr (Dehaspe and Toivonen 2001). The upgraded inductive logic programming systems will typically be more flexible than the systems it can emulate but typically also less efficient because there is a price to be paid for expressiveness. Finally, it may be possible to incorporate new features in the attribute-value learner by following the methodology. One feature that is often absent from propositional learners and may be easy to incorporate is the use of a background theory.

It should be mentioned that the methodology is not universal, that is, there exist also approaches, such as Muggleton's Progol (1995), which have directly been developed in first-order logic and for which no propositional counterpart exists. In such cases, however, it can be interesting to follow the inverse methodology, which would specialize the inductive logic programming system.

FOIL: An Illustration

One of the simplest and best-known inductive logic programming systems is FOIL (Quinlan 1990). It can be regarded as an upgrade of a rule learner such as CN2 (Clark and Niblett 1989). FOIL's problem setting is an instance of the learning from entailment setting introduced above (though it restricts the background theory to ground facts only and does not allow functors).

Like most rule-learning systems, FOIL employs a separate-and-conquer approach. It starts from the empty hypothesis, and then repeatedly searches for one rule that covers as many positive examples as possible and no negative example, adds it to the hypothesis, removes the positives covered by the rule, and then iterates. This process is continued until all positives are covered. To find one rule, it performs a hill-climbing search through the space of clauses ordered according to generality. The search starts at the most general rule, the one stating that all examples are positive, and then repeatedly specializes it. Among the specializations, it then selects the best one according to a heuristic evaluation based on information gain. A heuristic, based on the minimum description length principle, is then used to decide when to stop specializing clauses.

The key differences between FOIL and its propositional predecessors are the representation and the operators used to compute the specializations of a clause. It employs a refinement operator under θ-subsumption (Plotkin 1970) (see also ▶Logic of Generality). Such an operator essentially refines clauses by adding atoms to the condition part of the clause or applying substitutions to a clause. For instance, the clause

```
positive :- triangle(X),
            in(X,Y),
            color(X,C).
```

can be specialized to

```
positive :- triangle(X),
            in(X,Y),
            color(X,red).


positive :- triangle(X),
            in(X,Y),
            color(X,C),
            large(X).
positive :- triangle(X),
            in(X,Y),
            color(X,C),
            rectangle(Y).
  ...
```

The first specialization is obtained by substituting the variable `C` by the constant `red`, the other two by adding an atom (`large(X)`, `rectangle(Y)`, respectively) to the condition part of the rule. Inductive logic programming systems typically also employ syntactic restrictions – the so-called – that specify which clauses may be used in hypotheses. For instance, in the above example, the second argument of the `color` predicate belongs to the type *Color*, whereas the arguments of `in` are of type *Object* and consist of object identifiers.

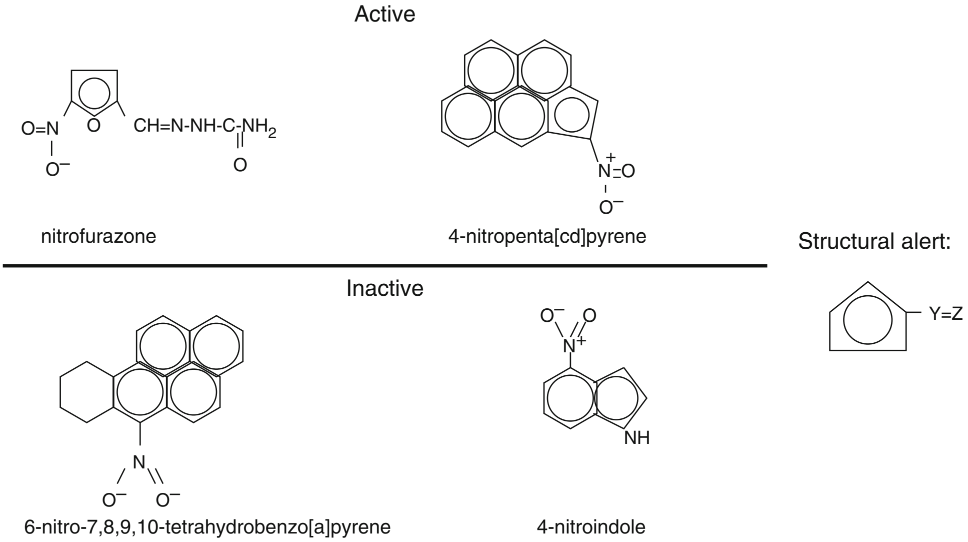
Application

Inductive logic programming has been successfully applied to many application domains, including bio- and chemo-informatics, ecology, network mining, software engineering, information retrieval, music analysis, web mining, natural language processing, toxicology, robotics, program synthesis, design, architecture, and many others. The best-known applications are in scientific domains. For instance, in structure–activity relationship prediction, one is given a set of molecules together with their activities, and background knowledge encoding functional groups, that is particular components of the molecule, and the task is to learn rules stating when a molecule is active or inactive. This is illustrated in Fig. 2 (after Srinivasan et al. 1996), where two molecules are active and two are inactive. One then has to find a pattern that discriminates the actives from the inactives. Structure–activity relationship (SAR) prediction is an essential step in, for instance, drug discovery. Using the general purpose inductive logic programming system Progol (Muggleton 1995) *structural alerts*, such as that shown in Fig. 2, have been discovered. These alerts allow one to distinguish the actives from the inactives – the one shown in the figure matches both of the actives but none of the inactives – and at the same time they are readily interpretable and provide useful insight into the factors determining the activity. To solve structure–activity relationship prediction problems using inductive logic programming, one must represent the molecules and hypotheses using the logical formalisms introduced above. The resulting representation is very similar to that employed in the Bongard problems: the objects are the atoms and relationships the bonds. Particular functional groups are encoded as background predicates.

State-of-the-Art

The upgrading methodology has been applied to a wide variety of machine learning systems and problems. There exist now inductive logic programming systems that:

- Induce logic programs from examples under various learning settings. This is by far the most popular class of inductive logic programming systems. Well-known systems include Aleph (Srinivasan 2007) and Progol (Muggleton 1995) as well as various variants of FOIL (Quinlan 1990). Some of these systems, especially Progol and Aleph, contain many features that are not present in propositional learning systems. Most of these systems focus



Inductive Logic Programming, Fig. 2 Predicting mutagenicity (Srinivasan et al. 1996) on a classification setting and learn the definition of a *single* predicate.

- Induce logical decision trees from examples. These are binary decision trees containing conjunctions of atoms (i.e., queries) as tests. If a query succeeds, then one branch is taken, else the other one. Decision tree methods for both classification and regression exist (see Blockeel and De Raedt 1998; Kramer and Widmer 2001).
- Mine for frequent queries, where queries are conjunctions of atoms. Such queries can be evaluated on an example. For instance, in the Bongard problem, the query `?- triangle (X), in (X, Y)` succeeds on the leftmost scenes and fails on the rightmost ones. Therefore, its frequency would be 6. The goal is then to find all queries that are frequent, that is, whose frequencies exceed a certain threshold. Frequent query mining upgrades the popular local pattern mining setting due to Agrawal et al. (1996) to inductive logic programming (see Dehaspe and Toivonen 2001).
- Learn or revise the definitions of theories, which consist of the definitions of multiple predicates, at the same time (cf. Wrobel 1996), and the entry in this encyclopedia. Several of these systems have their origin in the model inference system by Shapiro (1983) or the work by Angluin (1987).

Current Trends and Challenges

There are two major trends and challenges in inductive logic programming. The first challenge is to extend the inductive logic programming paradigm beyond the purely symbolic one. Important trends in this regard include:

- The combination of inductive logic programming principles with graphical and probabilistic models for reasoning about uncertainty. This is a field known as *statistical relational learning*, *probabilistic logic learning*, or *probabilistic inductive logic programming*. At the time of writing, this is a very popular research stream, attracting a lot of attention in the wider artificial intelligence community, cf. the entry ▶ Statistical Relational Learning in this encyclopedia. It has resulted in many relational or logical upgrades of well-known graphical models including Bayesian networks, Markov networks, hidden Markov models, and stochastic grammars.
- The use of relational distance measures for classification and clustering (Ramon and Bruynooghe 1998; Kirsten et al. 2001). These distances measure the similarity between two examples or clauses, while taking into account the underlying structure of the instances. These distances are then combined with standard classification and clustering methods such as k-nearest neighbor and k-means.
- The integration of relational or logical representations in reinforcement learning, known as ▶ Relational Reinforcement Learning (Dzeroski et al. 2001).

The power of inductive logic programming is also its weakness. The ability to represent complex objects and relations and the ability to make use of background knowledge add to the computational complexity. Therefore, a key challenge of inductive logic programming is tackling this added computational complexity. Even the simplest method for testing whether one hypothesis is more general than another – that is, θ-subsumption (Plotkin 1970) – is NP-complete. Similar tests are used for deciding whether a clause covers a particular example in systems such as FOIL. Therefore, inductive logic programming and relational learning systems are computationally much more expensive than their propositional counterparts. This is an instance of the expressiveness versus efficiency trade-off in computer science. Because of these computational difficulties, inductive logic programming has devoted a lot of attention to efficiency issues. On the theoretical side, there exist various results about the polynomial learnability of certain subclasses of logic programs (cf. Cohen and Page 1995, for an overview). From a practical perspective, there is quite some work on developing efficient methods for searching the hypothesis space and especially for evaluating the quality of hypotheses. Many of these methods employ optimized inference engines based on Prolog or database technology or constraint satisfaction methods (cf. Blockeel and Sebag 2003 for an overview).

Cross-References

▶ Multi-Relational Data Mining

Recommended Reading

A comprehensive introduction to inductive logic programming can be found in the book by De Raedt (2008) on logical and relational learning. Early surveys of inductive logic programming are contained in Muggleton and De Raedt (1994) and Lavrač and Džeroski (1994) and an account of its early history is provided in Sammut (1993). More recent collections on current trends can be found in the proceedings of the annual *Inductive Logic Programming Conference* (published in Springer's *Lectures Notes in Computer Science Series*) and special issues of the *Machine Learning Journal*. A summary of some key future challenges is given in Muggleton et al. (2012). An interesting collection of inductive logic programming and multi-relational data mining works are provided in Džeroski and Lavrač (2001). The upgrading methodology is described in detail in Van Laer and De Raedt (2001). More information on logical issues in inductive logic programming are given in the entry ▶ Logic of Generality in this encyclopedia, whereas the entries ▶ Statistical Relational Learning and ▶ Graph Mining are recommended for those interested in frameworks tackling similar problems using other types of representations.

Agrawal R, Mannila H, Srikant R, Toivonen H, Verkamo AI (1996) Fast discovery of association rules. In: Fayyad U, Piatetsky-Shapiro G, Smyth P, Uthurusamy R (eds) Advances in knowledge discovery and data mining. MIT Press, Cambridge, pp 307–328

Angluin D (1987) Queries and concept-learning. Mach Learn 2:319–342

Blockeel H, De Raedt L (1998) Top-down induction of first order logical decision trees. Artif Intell 101(1–2):285–297

Blockeel H, Sebag M (2003) Scalability and efficiency in multi-relational data mining. SIGKDD Explor 5(1):17–30

Bongard M (1970) Pattern recognition. Spartan Books, New York

Clark P, Niblett T (1989) The CN2 algorithm. Mach Learn 3(4):261–284

Cohen WW, Page D (1995) Polynomial learnability and inductive logic programming: methods and results. New Gener Comput 13:369–409

De Raedt L (2008) Logical and relational learning. Springer, Berlin

Dehaspe L, Toivonen H (2001) Discovery of relational association rules. In: Džeroski S, Lavrač N (eds) Relational data mining. Springer, Berlin/Heidelberg, pp 189–212

Džeroski S, De Raedt L, Driessens K (2001) Relational reinforcement learning. Mach Learn 43(1/2): 5–52

Džeroski S, Lavrač N (eds) (2001) Relational data mining. Springer, Berlin/New York

Kirsten M, Wrobel S, Horvath T (2001) Distance based approaches to relational learning and clustering. In: Džeroski S, Lavrač N (eds) Relational data mining. Springer, Berlin/Heidelberg, pp 213–232

Kramer S, Widmer G (2001) Inducing classification and regression trees in first order logic. In: Džeroski S, Lavrač N (eds) Relational data mining. Springer, Berlin/Heidelberg, pp 140–159

Lavrač N, Džeroski S (1994) Inductive logic programming: techniques and applications. Ellis Horwood, Chichester

Muggleton S (1995) Inverse entailment and Progol. New Gener Comput 13:245–286

Muggleton S, De Raedt L (1994) Inductive logic programming: theory and methods. J Log Program 19(20):629–679

Muggleton S, De Raedt L, Poole D, Bratko I, Flach P, Inoue K, Srinivasan A (2012) ILP Turns 20. Mach Learn 86:2–23

Plotkin GD (1970) A note on inductive generalization. In: Machine intelligence, vol 5. Edinburgh University Press, Edinburgh, pp 153–163

Quinlan JR (1990) Learning logical definitions from relations. Mach Learn 5:239–266

Ramon J, Bruynooghe M (1998) A framework for defining distances between first-order logic objects. In: Page D (ed) Proceedings of the eighth international conference on inductive logic programming. Lecture notes in artificial intelligence, vol 1446. Springer, Berlin/Heidelberg, pp 271–280

Sammut C (1993) The origins of inductive logic programming: a prehistoric tale. In: Muggleton S (ed) Proceedings of the third international workshop on inductive logic programming. J. Stefan Institute, Ljubljana, pp 127–148

Shapiro EY (1983) Algorithmic program debugging. MIT Press, Cambridge
Srinivasan A (2007) The Aleph Manual. http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph_toc.html
Srinivasan A, Muggleton S, Sternberg MJE, King RD (1996) Theories for mutagenicity: a study in first-order and feature-based induction. Artif Intell 85(1/2):277–299
Van Laer W, De Raedt L (2001) How to upgrade propositional learners to first order logic: a case study. In: Džeroski S, Lavrač N (eds) Relational data mining. Springer, Berlin/Heidelberg, pp 235–261
Wrobel S (1996) First-order theory refinement. In: De Raedt L (ed) Advances in inductive logic programming. Frontiers in artificial intelligence and applications, vol 32. IOS Press, Amsterdam, pp 14–33

Inductive Process Modeling

Ljupčo Todorovski
University of Ljubljana, Ljubljana, Slovenia

Synonyms

Process-based modeling

Definition

Inductive process modeling is a machine learning task that deals with the problem of learning quantitative *process models* from ▸ time series data about the behavior of an observed dynamic system. Process models are models based on ordinary differential equations that add an explanatory layer to the equations. Namely, scientists and engineers use models to both predict and explain the behavior of an observed system. In many domains, models commonly refer to processes that govern system dynamics and entities altered by those processes. Ordinary differential equations, often used to cast models of dynamic systems, offer one way to represent these mechanisms and can be used to simulate and predict the system behavior, but fail to make the processes and entities explicit. In response, process models tie the explanatory information about processes and entities to the mathematical formulation, based on equations, that enables simulation.

Inductive Process Modeling, Table 1 A process model of Predatory–Prey interaction between foxes and rabbits. The notation $d[X,t]$ indicates the time derivative of variable X

model predation;
entities fox{population}, rabbit{population};
process rabbit_growth;
entites rabbit;
*equations d[rabbit.conc,t] = 1.81 * rabbit.conc **
*(1 − 0.0003 * rabbit.conc);*
process fox_death;
entites fox;
*equations d[fox.conc,t] = −1.04 * fox.conc;*
process fox_rabbit_predation;
entities fox, rabbit;
equations
*d[fox.conc,t] = 0.03 * rabbit.conc * fox.conc;*
*d[rabbit.conc,t] = −1 * 0.3 * rabbit.conc * fox.conc;*

Table 1 shows a process model for a predator–prey interaction between foxes and rabbits. The three processes explain the dynamic change of the concentrations of both species (represented in the model as two *population* entities) through time. The *rabbit_growth* process states that the reproduction of rabbit is limited by the fixed environmental capacity. Similarly, the *fox_death* process specifies an unlimited exponential mortality function for the fox population. Finally, the *fox_rabbit_predation* process refers to the predator–prey interaction between foxes and rabbits that states that the prey concentration decreases and the predator one increases proportionally with the sizes of the two populations. The process model makes the structure of the model explicit and transparent to scientists; while at the same time it can be easily transformed in to a system of two differential equations by additively combining the equations for the time derivatives of the system variables *fox.conc* and *rabbit.conc*. Given initial values for these variables, one can simulate the equations to produce trajectories that correspond to the population dynamics through time.

The processes from Table 1 instantiate more general generic processes, that can be used for modeling any ecological system. For example: is a general form of the *fox_rabbit_predation* process from the example model in Table 1. Note that in the generic process, the parameters are replaced with numeric ranges and the entities with identifiers of generic entities (i.e., *Predator* and *Prey* are identifiers that refer to instances of the generic entity *population*).

```
generic process predation;
  entities Predator{population}, Prey{population};
  parameters ar[0.01, 10], ef[0.001, 0.8];
  equations
         d[Predator.conc,t] = ef * ar * Prey.conc *
Predator.conc;
       d[Prey.conc,t] = −1 * ar * Prey.conc * Preda-
tor.conc;
```

Having defined entities and processes on an example, one can define the task of inductive process modeling as: Given

- Time series observations for a set of numeric system variables as they change through time
- A set of entities that the model might include
- Generic processes that specify casual relations among entities
- Constraints that determine plausible relations among processes and entities in the model

Find a specific process model that explains the observed data and the simulation of which closely matches observed time series.

There are two approaches for solving the task of inductive process modeling. The first is the transformational approach that transforms the given knowledge about entities, processes, and constraints to ▸ language bias for equation discovery and uses the Lagramge method for ▸ equation discovery in turn (Todorovski and Džeroski 1997, 2007). The second approach performs search through the space of candidate process models to find the one that matches the given time series data best.

Inductive process modeling methods IPM (Bridewell et al. 2008) and HIPM (Todorovski et al. 2005) follow the second approach. IPM is a naïve method that exhaustively searches the space of candidate process models following the ▸ learning as search paradigm. The search space of candidate process models is defined by the sets of generic processes and of entities in the observed system specified by the user. IPM first matches the type of each entity against the types of entities involved in each generic process and produces a list of all possible instances of that generic process. For example, the generic process *predation*, from the example above, given two *population* entities *fox* and *rabbit*, can be instantiated in four different ways (*fox_fox_predation*, *fox_rabbit_predation*, *rabbit_fox_predation*, and *rabbit_rabbit_predation*). The IPM search procedure collects the set of all possible instances of all the generic processes and uses them as a set of candidate model components. In the search phase, all combinations of these model components are being matched against observed ▸ time series. The matching involves the employment of gradient-descent methods for nonlinear optimization to estimate the optimal values of the process model parameters. As output, IPM reports the process models with the best match.

Trying out all components' combinations is prohibitive in many situations since it obviously leads to combinatorial explosion. HIPM employs constraints that limit the space of combinations by ruling-out implausible or forbidden combinations. Examples of such constraints in the predator–prey example above include rules that a proper process model of population dynamics should include a single growth and a single mortality process per species, the predator–prey process should relate two different species, and different predator–prey interaction should refer to different population pairs. HIPM specifies the rules in a hierarchy of generic processes where each node in the hierarchy specifies a rule for proper combination/selection of process instances.

Cross-References

▸ Equation Discovery

Recommended Reading

Bridewell W, Langley P, Todorovski L, Džeroski S (2008) Inductive process modeling. Mach Learn 71(1):1–32

Todorovski L, Džeroski S (1997) Declarative bias in equation discovery. In: Fisher DH (ed) Proceedings of the fourteenth international conference on machine learning,Nashville

Todorovski L, Džeroski S (2007) Integrating domain knowledge in equation discovery. In: Džeroski S, Todorovski L (eds) Computational discovery of scientific knowledge. LNCS, vol 4660. Springer, Berlin

Todorovski L, Bridewell W, Shiran O, Langley P (2005) Inducing hierarchical process models in dynamic domains. In: Veloso MM, Kambhampati S (eds) Proceedings of the twentieth national conference on artificial intelligence, Pittsburgh

Inductive Program Synthesis

► Inductive Programming

Inductive Programming

Pierre Flener[1] and Ute Schmid[2]
[1]Department of Information Technology,
Uppsala University, Uppsala, Sweden
[2]Faculty of Information Systems and Applied
Computer Science, University of Bamberg,
Bamberg, Germany

Abstract

Inductive programming is introduced as a branch of program synthesis which is based on inductive inferece where recursive, declarative programs are constructed from incomplete specifications, especially from input/output examples. Inductive logic programming as well as inductive functional programming are addressed. Central concepts such as predicate invention and background knowledge are defined. Two worked-out examples are presented to illustrate inductive logic as well as inductive functional programming.

Synonyms

Example-based programming; Inductive program synthesis; Inductive synthesis; Programming by examples; Program synthesis from examples

Definition

Inductive programming is the inference of an algorithm or program featuring recursive calls or repetition control structures, starting from information that is known to be incomplete, called the *evidence*, such as positive and negative input-output examples or clausal constraints. The inferred program must be correct with respect to the provided evidence, in a **generalization** sense: it should be neither equivalent to it nor inconsistent. Inductive programming is guided explicitly or implicitly by a **language bias** and a **search bias**. The inference may draw on background knowledge or query an oracle. In addition to **induction**, **abduction** may be used. The restriction to algorithms and programs featuring recursive calls or repetition control structures distinguishes inductive programming from **concept learning** or **classification**.

We here restrict ourselves to the inference of declarative programs, whether functional or logic, and dispense with repetition control structures in the inferred program in favor of recursive calls.

Motivation and Background

Inductive program synthesis is a branch of the field of *program synthesis*, which addresses a cognitive question as old as computers, namely, the understanding of the human act of computer

Most of the work by this author was done while on leave of absence in 2006/07 as a Visiting Faculty Member and Erasmus Exchange Teacher at Sabancı University, Turkey.

programming, to the point where a computer can be made to help in this task (and ultimately to enhance itself). See Flener (2002) and Gulwani et al. (2014) for surveys; the other main branches of program synthesis are based on deductive inference, namely, *constructive program synthesis* and *transformational program synthesis*. In such *deductive program synthesis*, the provided information, called the *specification*, is assumed to be complete (in contrast to inductive program synthesis where the provided information is known to be incomplete), and the presence of repetitive or recursive control structures in the synthesized program is not imposed.

Research on the inductive synthesis of recursive *functional* programs started in the early 1970s and was brought onto firm theoretical foundations with the seminal THESYS system of Summers (1977) and work of Biermann (1978), where all the evidence is handled non-**incrementally**. Essentially, the idea is first to infer computation *traces* from input-output examples (**instances**) and then to use a **trace-based programming** method to fold these traces into a recursive program. The main results until the mid-1980s were surveyed in Smith (1984). Due to limited progress with respect to the range of programs that could be synthesized, research activities decreased significantly in the next decades. However, a new approach that formalizes functional program synthesis in the term rewriting framework and that allows the synthesis of a broader class of programs than the classical approaches is pursued in Kitzelmann and Schmid (2006).

The advent of *logic* programming brought a new lan but also a new direction in the early 1980s, especially due to the MIS system of Shapiro (1983), eventually spawning the new field of **inductive logic programming** (ILP). Most of this ILP work addresses a wider class of problems, as the focus is *not* only on recursive logic programs: more adequate designations are inductive **theory revision** and *declarative program debugging*, as an additional input is a possibly empty initial theory or program that is **incrementally** revised or debugged according to each newly presented piece of evidence, possibly in the presence of background knowledge or an oracle. The main results on the inductive synthesis of recursive logic programs were surveyed in Flener and Yılmaz (1999).

Structure of Learning System

The core of an inductive programming system is a mechanism for constructing a recursive **generalization** for a set of input/output examples (**instances**). Although we use the vocabulary of logic programming, this method also covers the synthesis of functional programs.

The input, often a set of input/output examples, is called the *evidence*. Further evidence may be queried from an *oracle*. Additional information, in the form of predicate symbols that can be used during the synthesis, can be provided as *background knowledge*. Since the **hypothesis space** – the set of legal recursive programs – is infinite, a **language bias** is introduced. One particularly useful and common approach in inductive programming is to provide a statement bias by means of a *program schema*.

The evidential synthesis of a recursive program starts from the provided evidence for some predicate symbol and works essentially as follows. A program schema is chosen to provide a template for the program structure, where all yet undefined predicate symbols must be instantiated during the synthesis. Predefined predicate symbols of the background knowledge are then chosen for some of these undefined predicate symbols in the template. If it is deemed that the remaining undefined predicate symbols cannot all be instantiated via purely structural generalization by non-recursive definitions, then the method is recursively called to infer recursive definitions for some of them (this is called **predicate invention** and amounts to shifting the *vocabulary bias*); otherwise the synthesis ends successfully right away. This generic method can backtrack to any choice point for synthesizing alternative programs.

In the rest of this section, we discuss this basic terminology of inductive programming more precisely. In the next section, instantiations of this generic method by some well-known methods are presented.

The Evidence and the Oracle

The evidence is often limited to ground positive examples of the predicate symbols that are to be defined. Ground negative examples are convenient to prevent overgeneralization, but should be used constructively and not just to reject candidate programs. A useful generalization of ground examples is evidence in the form of a set of (nonrecursive) clauses, as variables and additional predicate symbols can then be used.

Example 1 The $delOdds(L, R)$ relation, which holds if and only if R is the integer list L without its odd elements, can be incompletely described by the following clausal evidence:

$$\begin{aligned} delOdds([\,],[\,]) &\leftarrow true \\ delOdds([X],[\,]) &\leftarrow odd(X) \\ delOdds([X],[X]) &\leftarrow \neg odd(X) \\ delOdds([X,Y],[Y]) &\leftarrow odd(X),\ \neg odd(Y) \\ delOdds([X,Y],[X,Y]) &\leftarrow \neg odd(X),\ \neg odd(Y) \\ false &\leftarrow delOdds([X],[X]),\ odd(X) \end{aligned} \tag{1}$$

The first clause is a ground positive example, whereas the second and third clauses generalize the infinity of ground positive examples, such as $delOdds([5],[\,])$ and $delOdds([22],[22])$, for handling singleton lists, while the fourth and fifth clauses summarize the infinity of ground positive examples for handling lists of two elements, the second one being even: these clauses make explicit the underlying filtering relation (*odd*) that is *intrinsic* to the problem at hand but cannot be provided via ground examples and would otherwise have to be guessed. The sixth clause summarizes an infinity of ground negative examples for handling singleton lists, namely, where the only element of the list is odd but not filtered.

In some methods, especially for the induction of functional programs, the first *n* positive input-output examples with respect to the underlying data type are presented (e.g., for linear lists, what to do with the empty list, with a one-element list, up to a list with three elements); because of this ordering of examples, no explicit presentation of negative examples is then necessary.

Inductive program synthesis should be monotonic in the evidence (more evidence should never yield a less complete program, and less evidence should not yield a more complete program) and should not be sensitive to the order of presentation of the evidence.

Program Schemas

Informally, a *program schema* contains a template program and a set of axioms. The *template* abstracts a class of actual programs, called *instances*, in the sense that it represents their dataflow and control flow by means of placeholders, but does not make explicit all their actual computations nor all their actual data structures. The *axioms* restrict the possible instances of the placeholders and define their interrelationships. Note that a schema is problem independent. Let us here take a **first-order logic** approach and consider templates as open logic programs (i.e. programs where some placeholder predicate symbols are left undefined or *open*; a program with no open predicate symbols is said to be *closed*) and axioms as first-order specifications of these open predicate symbols.

Example 2 Most methods of inductive synthesis are biased by program schemas whose templates have clauses of the forms in the following generic template:

$$\begin{aligned} r(X,Y,Z) \leftarrow\ & c(X,Y,Z), \\ & p(X,Y,Z) \\ r(X,Y,Z) \leftarrow\ & d(X,H,X_1,\ldots,X_t,Z), \\ & r(X_1,Y_1,Z),\ \ldots,\ r(X_t,Y_t,Z), \\ & q(H,Y_1,\ldots,Y_t,Z,Y) \end{aligned} \tag{2}$$

where c, d, p, q are open predicate symbols, X is a nonempty sequence of terms, and Y, Z are possibly empty sequences of terms. The intended semantics of this generic template can be informally described as follows. For an arbitrary relation r over parameters X, Y, Z, an instance of this generic template is to determine the values of *result parameter* Y corresponding to a given value of *induction parameter* X, considering the value of *auxiliary parameter* Z. Two cases arise: either the c test succeeds and X has a value for which Y can be easily directly computed through p, or X has a value for which Y cannot be so easily directly computed and the *divide-and-conquer principle* is applied:

1. *divide* X through d into a term H and t terms $X_1, \ldots, X_t$ of the same type as X but smaller than X according to some well-founded relation;
2. *conquer* through t recursive calls to r to determine the values of $Y_1, \ldots, Y_t$ corresponding to $X_1, \ldots, X_t$, respectively, considering the value of Z;
3. *combine* through q the terms $H, Y_1, \ldots, Y_t, Z$ to build Y.

Enforcing this intended semantics must be done manually, as any instance template by itself has no semantics, in the sense that any program is an instance of it (it suffices to define c by a program that always succeeds and p by the given program). One way to do this is to attach to a template some axioms (see Smith (1985) for the divide-and-conquer axioms), namely, the set of specifications of its open predicate symbols: these specifications refer to each other, including the one of r, and are generic (because even the specification of r is unknown), but can be manually abduced once and for all according to the informal semantics of the schema.

Predicate Invention

Another important **language bias** is the available vocabulary, which is here the set of predicate symbols mentioned in the evidence set or actually defined in the background knowledge (and possibly mentioned by the oracle). If an inductive synthesis fails, other than backtracking to a different program schema (i.e., shifting the statement bias), one can try and shift the vocabulary bias by inventing new predicate symbols and inducing programs for them in the extended vocabulary; this is also known as performing **constructive induction**. Only the invention of recursively defined predicate symbols is *necessary*, as a non-recursive definition of a predicate symbol can be eliminated by substitution (under **resolution**) for its calls in the induced program (even though that might make the program longer).

In general, it is undecidable whether **predicate invention** is necessary to induce a finite program in the vocabulary of its evidence and background knowledge (as a consequence of Rice's theorem, 1953), but introducing new predicate symbols always allows the induction of a finite program (as a consequence of a result by Kleene), as shown in Stahl (1995). The necessity of shifting the vocabulary bias is only decidable for some restricted languages (but the bias shift attempt might then be unsuccessful), so in practice one often has to resort to heuristics. Note that an inductive synthesizer of recursive algorithms may be recursive itself: it may recursively invoke itself for a necessary new predicate symbol.

Other than the decision problem, the difficulties of predicate invention are as follows. First, adequate formal parameters for a new predicate symbol have to be identified among all the variables in the clause using it. This can be done instantaneously by using precomputations done manually once and for all at the template level. Second, evidence for a new predicate symbol has to be **abduced** from the current program using the evidence for the old predicate symbol. This usually requires an oracle for the old predicate symbol, whose program is still unfinished at that moment and cannot be used. Third, the abduced evidence may be less numerous than for the old predicate symbol (note that if the new predicate symbol is in a recursive clause, then no new evidence might be abduced from the old evidence that is covered by the base clauses) and can be quite sparse, so that the new synthesis is more difficult. This *sparseness problem* can be illustrated by an example.

Example 3 Given the positive ground examples $\mathit{factorial}(0,1)$, $\mathit{factorial}(1,1)$, $\mathit{factorial}(2,2)$, $\mathit{factorial}(3,6)$, and $\mathit{factorial}(4,24)$ and given the still open program:

$$\begin{aligned}
factorial(N,F) &\leftarrow N=0,\ F=1\\
factorial(N,F) &\leftarrow add(M,1,N),\\
&\qquad factorial(M,G),\\
&\qquad product(N,G,F)
\end{aligned}$$

where *add* is known but *product* was just invented (and named so only for the reader's convenience), the abduceable examples are *product*$(1,1,1)$, *product*$(2,1,2)$, *product*$(3,2,6)$, and *product* $(4,6,24)$, which is hardly enough for inducing a recursive program for *product*; note that there is one less example than for *factorial*. Indeed, examples such as *product*$(3,6,18)$, *product*$(2,6,12)$, *product*$(1,6,6)$, etc. are missing, which puts the given examples more than one resolution step apart, if not on different **resolution** paths. This is aggravated by the absence of an oracle for the invented predicate symbol, which is not necessarily intrinsic to the task at hand (although *product* actually is intrinsic to] *factorial*).

Background Knowledge

In an inductive programming context, background knowledge is particularly important, as the inference of recursive programs is more difficult than the inference of **classifiers**. For the efficiency of synthesis, it is crucial that this collection of definitions of the predefined predicate symbols be *annotated* with information about the *types* of their arguments and about whether some *well-founded relation* is being enforced between some of their arguments, so that semantically suitable instances for the open predicate symbols of any chosen program schema can be readily spotted. (This requires in turn that the types of the arguments of the predicate symbols in the provided evidence are declared as well.) The background knowledge should be problem independent, and an inductive programming method should be able to perform *knowledge mobilization*, namely organizing it dynamically according to relevance to the current task.

In data-driven, analytical approaches, background knowledge is used in combination with **explanation-based learning** (EBL) methods, such as **abduction** (see Example 4) or systematic rewriting of input/output examples into computational traces (see Example 5).

Background knowledge can also be given in the form of constraints or an explicit inductive bias as in meta-interpretative learning (Muggleton and Lin 2013) or in using higher-order patterns (Katayama 2006).

Programs and Data

Example 4 The DIALOGS (Dialogue-based Inductive-Abductive LOGic program Synthesizer) method (Flener 1997) is interactive. The main design objective was to take all extra burden from the specifier by having the method ask for exactly and only the information it needs, default answers being provided wherever possible. As a result, no evidence needs to be prepared in advance, as the method invents its own candidate evidence and queries the oracle about it, with an opportunity to declare (at the oracle/specifier's risk) that enough information has been provided. All answers by the oracle are stored as *judgments*, to prevent asking the same query twice. This is suitable for all levels of expertise of human users, as the queries are formulated in the specifier's initially unknown conceptual language, in a way such that the specifier must know the answers if she really feels the need for the wanted program. The method is schema-biased, and the current implementation has two schemas. The template of the *divide-and-conquer* schema has the generality of the generic template (2). The template of the *accumulate* schema extends this by requiring an accumulator in the sequence Z of auxiliary parameters. The evidence language (**observation language**) is (non-recursive) logic programs with negation. Type declarations are provided as a **language bias**. The program language (**hypothesis language**) is recursive logic programs with negation, with possibly multiple base cases and recursive cases.

For instance, starting from the empty program for the relation *delOdds* in Example 1, the algorithm design choices of using the *divide-*

and-conquer schema with R as result parameter and L as induction parameter decomposed (with $t = 1$) through head-tail decomposition by d lead to the following intermediate open program:

$$\begin{aligned} delOdds(L,R) &\leftarrow c(L,R),\ p(L,R) \\ delOdds(L,R) &\leftarrow d(L,H,L_1), \\ &\quad delOdds(L_1,R_1), \\ &\quad q(H,R_1,R) \\ c(\_,\_) &\leftarrow true \\ d(L,H,T) &\leftarrow L = [H|T] \end{aligned} \tag{3}$$

The first five evidential clauses for *delOdds* in (1) are then implicitly interactively acquired from the oracle/specifier by the following question and answer dialogue, leading the specifier to reveal the intrinsic predicate symbol *odd*:

DIALOGS: When does *delOdds*([ ], R) hold?
Specifier: If $R = [\,]$.
DIALOGS: When does *delOdds*([X], R) hold?
Specifier: If ($odd(X)$ *and* $R = [\,]$) *or* ($\neg odd(X)$ *and* $R = [X]$).
DIALOGS: When does *delOdds*([X, Y], R) hold, assuming $odd(Y)$?
Specifier: If ($odd(X)$ *and* $R = [\,]$) *or* ($\neg odd(X)$ *and* $R = [X]$).
DIALOGS: When does *delOdds*([X, Y], R) hold, assuming $\neg odd(Y)$?
Specifier: If ($odd(X)$ *and* $R = [Y]$) *or* ($\neg odd(X)$ *and* $R = [X, Y]$).

Next, **abduction** infers the following evidence set for the still open predicate symbols p and q:

$$\begin{aligned} p([\,],[\,]) &\leftarrow true \\ p([X],[\,]) &\leftarrow odd(X) \\ p([X],[X]) &\leftarrow \neg odd(X) \\ p([X,Y],[Y]) &\leftarrow odd(X),\ \neg odd(Y) \\ p([X,Y],[X,Y]) &\leftarrow \neg odd(X),\ \neg odd(Y) \\ q(X,[\,],[\,]) &\leftarrow odd(X) \\ q(X,[\,],[X]) &\leftarrow \neg odd(X) \\ q(X,[Y],[Y]) &\leftarrow odd(X) \\ q(X,[Y],[X,Y]) &\leftarrow \neg odd(X) \end{aligned}$$

From this, induction infers the following closed programs for p and q:

$$\begin{aligned} p([\,],[\,]) &\leftarrow true \\ q(H,L,[H|L]) &\leftarrow \neg odd(H) \\ q(H,L,L) &\leftarrow odd(H) \end{aligned} \tag{4}$$

The final closed program is the union of the programs (3) and (4), as no predicate invention is deemed necessary. Sample syntheses with predicate invention are presented in Flener (1997) and Flener and Yılmaz (1999).

Example 5 The THESYS method (Summers 1977) was one of the first methods for the inductive synthesis of functional (Lisp) programs. Although it has a rather restricted scope, it can be seen as the methodological foundation of many later methods for inducing functional programs. The noninteractive method is schema biased, and the implementation has two schemas. Upon adaptation to functional programming, the template of the *linear recursion* schema is the instance of the generic template (2) obtained by having X as a sequence of exactly one induction parameter and Z as the empty sequence of auxiliary parameters, and by dividing X into $t = 1$ smaller value X_t, so that there is only $t = 1$ recursive call. The template of the *accumulate* schema extends this by having Z as a sequence of exactly one auxiliary parameter, playing the role of an accumulator. The evidence language (**observation language**) is sets of ground positive examples. The program language (**hypothesis language**) is recursive functional programs, with possibly multiple base cases, but only one recursive case. The only primitive functions are *nil*, *cons*, *head*, *tail*, and *empty*, because the implementation is limited to the list data type, inductively defined by $list \equiv nil \mid cons(x, list)$, under the axioms $empty(nil) = true$, $head(cons(x,y)) = x$, and $tail(cons(x,y)) = y$. There is no function invention.

For instance, from the following examples of a list unpacking function:

$$\begin{aligned}
&unpack(nil) &&= nil\\
&unpack((A)) &&= ((A))\\
&unpack((A\ B)) &&= ((A)\ (B))\\
&unpack((A\ B\ C)) &&= ((A)\ (B)\ (C))
\end{aligned}$$

the **abduced** traces are:

$$\begin{aligned}
&empty(X) &&\rightarrow nil\\
&empty(tail(X)) &&\rightarrow cons(X, nil)\\
&empty(tail(tail(X))) &&\rightarrow cons(cons(head(X), nil), cons(tail(X), nil))\\
&empty(tail(tail(tail(X)))) &&\rightarrow cons(cons(head(X), nil), cons(cons(head(tail(X)), nil),\\
& &&\quad\ cons(tail(tail(X)), nil)))
\end{aligned}$$

and the **induced** program is:

$$\begin{aligned}
unpack(X) = \ &empty(X) &&\rightarrow nil,\\
&empty(tail(X)) &&\rightarrow cons(X, nil),\\
&true &&\rightarrow cons(cons(head(X), nil), unpack(tail(X)))
\end{aligned}$$

A modern extension of THESYS is the IGOR method (Kitzelmann and Schmid 2006). The underlying program template describes the set of all functional programs with the following restrictions: built-in functions can only be first-order, and no nested or mutual recursion is allowed. IGOR adopts the two-step approach of THESYS. Synthesis is still restricted to structural problems, where only the structure of the arguments matters, but not their contents, such as in list reversing. Nevertheless, the scope of synthesizable programs is considerably larger. For instance, tree-recursive functions and functions with hidden parameters can be induced. Most notably, programs consisting of a calling function and an arbitrary set of further recursive functions can be induced. The first step of synthesis (trace construction) is therefore expanded such that traces can contain nestings of conditions. The second step is expanded such that the synthesis of a function can rely on the invention and synthesis of other functions (i.e., IGOR uses a technique of function invention in correspondence to the concept of **predicate invention** introduced above). An extension, IGOR2, relies on constructor term rewriting techniques. The two synthesis steps are merged into one and make use of background knowledge. Therefore, the synthesis of programs for semantic problems, such as list sorting, becomes feasible.

Applications

In the framework of *software engineering*, inductive programming is defined as the inference of information that is pertinent to the construction of a generalized computational system for which the provided evidence is a representative sample (Flener and Partridge 2001). In other words, inductive programming does *not* have to be a panacea for software development in the large and infer a complete software system in order to be useful: it suffices to induce, for instance, a self-contained system module while programming in the small, problem features and decision logic for specification acquisition and enhancement or support for debugging and testing. Inductive programming is then not always limited to programs with repetitive or recursive control structures. There are opportunities for synergy with manual programming and deductive program synthesis, as there are sometimes system modules that no one knows how to specify in a complete way, or that are harder to specify or program in a complete way, and yet where incomplete information such as input-output examples is readily available. More examples and pointers to the literature are given in Flener (2002, Section 5) and Flener and Partridge (2001).

In the context of *end-user programming*, inductive programming methods can be used to enable nonexpert users to take advantage of the more sophisticated functionalities offered by their software. This kind of application is in the focus of **programming by demonstration** (PBD).

Finally, it is worth having an evidential synthesizer of recursive algorithms invoked by a more general-purpose machine learning method when necessary predicate invention is detected or conjectured, as such general methods require a lot of evidence to infer reliably a recursively defined hypothesis.

Future Directions

Inductive programming is still mainly a topic of basic research, exploring how the intellectual ability of humans to infer generalized recursive procedures from incomplete evidence can be captured in the form of synthesis methods. Already a variety of promising methods are available. A necessary step should be to compare and analyze the current methods. A first extensive comparison of different ILP methods for inductive programming was presented some years ago (Flener and Yılmaz 1999). An up-to-date analysis should take into account not only ILP methods but also methods for the synthesis of functional programs, using classical (Kitzelmann and Schmid 2006) as well as evolutionary (Olsson 1995) methods. The methods should be compared with respect to the required quantity of evidence, the kind and amount of background knowledge, the scope of programs that can be synthesized, and the efficiency of synthesis. Such an empirical comparison should result in the definition of characteristics that describe concisely the scope, usefulness, and efficiency of the existing methods in different problem domains. A first step toward such a systematic comparison was presented in Hofmann et al. (2009).

Since only a few inductive programming methods can deal with semantic problems, it should be useful to investigate how inductive programming methods can be combined with other machine learning methods, such as kernel-based **classification**.

Finally, the existing methods should be adapted to a broad variety of application areas in the context of programming assistance, as well as in other domains where recursive data structures or recursive procedures are relevant.

Cross-References

▸ Explanation-Based Learning
▸ Inductive Logic Programming
▸ Programming by Demonstration
▸ Programming by Example (PBE)
▸ Trace-Based Programming

Recommended Reading

- Online Platform of the Inductive Programming Community: http://www.inductive-programming.org/.
- Journal of *Automated Software Engineering*, *Special Issue on Inductive Programming*, April 2001: Flener and Partridge (2001), http://user.it.uu.se/~pierref/ase/.
- *Biannual Workshops on Approaches and Applications of Inductive Programming*: http://www.cogsys.wiai.uni-bamberg.de/aaip/.
- *Journal of Machine Learning Research*, *Special Topic on Approaches and Applications of Inductive Programming*, February/March 2006: http://jmlr.csail.mit.edu/papers/topic/inductive_programming.html.
- *Dagstuhl Report 3/12 on Approaches and Applications of Inductive Programming* http://drops.dagstuhl.de/opus/volltexte/2014/4507/.

Biermann AW (1978) The inference of regular LISP programs from examples. IEEE Trans Syst Man Cybern 8(8):585–600

Flener P (1997) Inductive logic program synthesis with DIALOGS. In: Muggleton SH (ed) Revised selected papers of the 6th international workshop on inductive logic programming (ILP 1996), Stockholm. Volume 1314 of lecture notes in artificial intelligence. Springer, pp 175–198

Flener P (2002) Achievements and prospects of program synthesis. In: Kakas A, Sadri F (eds) Computational logic: logic programming and beyond; essays in honour of Robert A. Kowalski. Volume 2407 of lecture notes in artificial intelligence. Springer, Berlin/New York, pp 310–346

Flener P, Partridge D (2001) Inductive programming. Autom Softw Eng 8(2):131–137

Flener P, Yılmaz S (1999) Inductive synthesis of recursive logic programs: achievements and prospects. J Log Program 41(2–3):141–195

Gulwani S, Kitzelmann E, Schmid U (2014) Approaches and Applications of Inductive Programming (Dagstuhl Seminar 13502). Dagstuhl Reports 3/12, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl

Hofmann M, Kitzelmann E, Schmid U (2009) A unifying framework for analysis and evaluation of inductive programming systems. In: Goerzel B, Hitzler P, Hutter M (eds) Proceedings of the second conference on artificial general intelligence (AGI-09, Arlington, Virginia, 6–9 March 2009), Amsterdam. Atlantis Press, pp 55–60

Katayama S (2005) Systematic search for lambda expressions. In: Trends in functional programming. Intellect, Bristol, pp 111–126

Kitzelmann E, Schmid U (2006) Inductive synthesis of functional programs – an explanation based generalization approach. J Mach Learn Res 7(Feb): 429–454

Muggleton SH, Lin D (2013) Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. In: Rossi F (ed) IJCAI 2013, proceedings of the 23rd international joint conference on artificial intelligence, Beijing, 3–9 Aug 2013. IJCAI/AAAI, pp 1551–1557

Olsson JR (1995) Inductive functional programming using incremental program transformation. Artif Intell 74(1):55–83

Shapiro EY (1983) Algorithmic program debugging. The MIT Press, Cambridge

Smith DR (1984) The synthesis of LISP programs from examples: a survey. In: Biermann AW, Guiho G, Kodratoff Y (eds) Automatic program construction techniques. Macmillan, New York, pp 307–324

Smith DR (1985) Top-down synthesis of divide-and-conquer algorithms. Artificial Intelligence, 27(1):43–96

Stahl I (1995) The appropriateness of predicate invention as bias shift operation in ILP. Mach Learn 20(1–2):95–117

Summers PD (1977) A methodology for LISP program construction from examples. J ACM 24(1): 161–175

Inductive Synthesis

▸ Inductive Programming

Inductive Transfer

Ricardo Vilalta[1], Christophe Giraud-Carrier[2], Pavel Brazdil[3], and Carlos Soares[3,4]
[1]Department of Computer Science, University of Houston, Houston, TX, USA
[2]Department of Computer Science, Brigham Young University, Provo, UT, USA
[3]LIAAD-INESC Tec/Faculdade de Economia, University of Porto, Porto, Portugal
[4]LIAAD-INESC Porto L.A./Faculdade de Economia, University of Porto, Porto, Portugal

Abstract

We describe different scenarios where a learning mechanism is capable of acquiring experience on a source task, and subsequently exploit such experience on a target task. The core ideas behind this ability to transfer knowledge from one task to another have been studied in the machine learning literature under different titles and perspectives. Here we describe some of them under the names of inductive transfer, transfer learning, multitask learning, metasearching, meta-generalization, and domain adaptation.

Synonyms

Domain adaptation; Multitask learning; Transfer learning; Transfer of knowledge across domains

Definition

Inductive transfer refers to the ability of a learning mechanism to improve performance on the current or *target* task after having learned a different but related concept or skill on a previous *source* task. Transfer may additionally occur between two or more learning tasks that are being undertaken concurrently. The object being transferred may refer to instances, features, a particular form of search bias, an action policy, background knowledge, etc.

Motivation and Background

Learning is not the result of an isolated task that starts from scratch with every new problem. Instead, a learning algorithm should exhibit the ability to adapt through a mechanism dedicated to transfer knowledge gathered from previous experience. The problem of transfer of knowledge is central to the field of machine learning and is also known as *inductive transfer*. In this case, knowledge can be understood as a collection of patterns observed across tasks. One view of the nature of patterns across tasks is that of invariant transformations. For example, image recognition of a target object is simplified if the object is invariant under rotation, translation, scaling, etc. A learning system should be able to recognize a target object on an image even if previous images show the object in different sizes or from different angles. Hence, inductive transfer studies know how to improve learning by detecting, extracting, and exploiting (meta)knowledge in the form of invariant transformations across tasks.

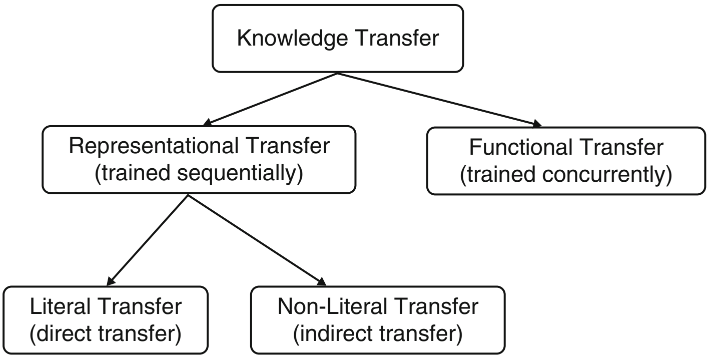
Similarly, in competitive games involving teams of robots (e.g., RoboCup Soccer), transferring knowledge learned from one task to another task is crucial to acquire skills necessary to beat the opponent team. Specifically, imagine a situation where a team of robots has been taught to keep a soccer ball away from the opponent team. To achieve that goal, robots must learn to keep the ball, pass the ball to a close teammate, etc., always trying to remain at a safe distance from the opponents. Now let us assume that we wish to teach the same team of robots to be efficient at scoring against a team of defending robots. Knowledge gained during the first activity can be transferred to the second one. Specifically, a robot can prefer to perform an action learned in the past over actions proposed during the current task, because the past action has a significant higher merit value. For example, a robot under the second task may learn to recognize that it is preferable to shoot than to pass the ball because the goal is very close. This action can be learned from the first task by recognizing that the precision of a pass is contingent upon the proximity of the teammate.

Structure of the Learning System

The main idea behind a learning architecture using knowledge transfer is to produce a source model from which knowledge can be extracted and transferred to a target model. This allows for multiple scenarios (Brazdil et al. 2009; Pratt and Thrun 1997). For example, the target and source models can be trained at different times in such a way that the transfer takes place after the source model has been trained. In this case there is an explicit form of knowledge transfer, also called *representational transfer*. In contrast, we use the term *functional transfer* to denote the case where two or more models are trained simultaneously; in this case the models share (part of) their internal structure during learning (see Neural Networks below). Under representational transfer, we denote as *literal transfer* the case when the source model is left intact and as *nonliteral transfer* the case when the source model is modified before knowledge is transferred to the target model. In nonliteral transfer some processing takes place on the source model before it is used to initialize the target model (see Fig. 1).

Neural Networks. A learning paradigm amenable to test the feasibility of knowledge transfer is that of neural networks (Caruana 1993). A popular form of (functional) knowledge transfer is effected through multitask learning, where the output nodes in the multilayer network represent more than one task. In such a scenario, internal nodes are shared by different tasks dynamically during learning. As an illustration, consider the problem of learning to classify astronomical objects from images mapping

Inductive Transfer, Fig. 1 A taxonomy of inductive transfer

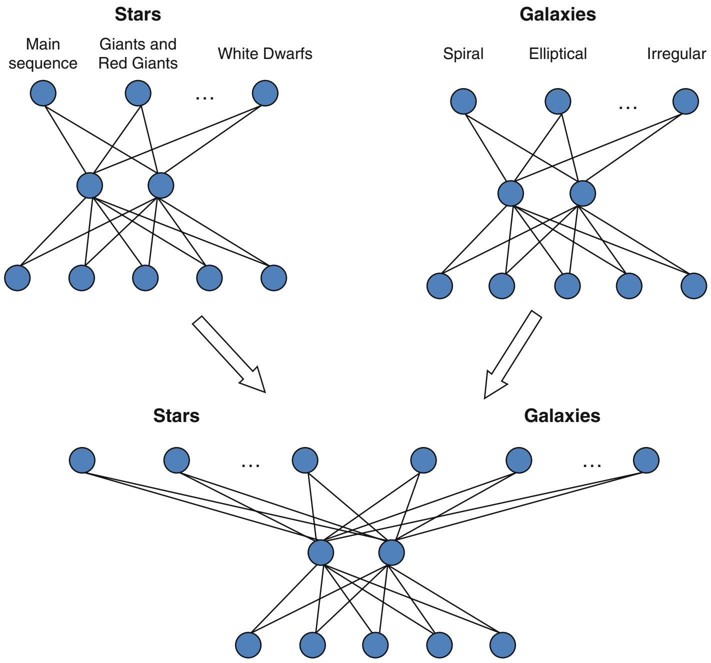

Knowledge Transfer
Representational Transfer (trained sequentially)
Functional Transfer (trained concurrently)
Literal Transfer (direct transfer)
Non-Literal Transfer (indirect transfer)

the sky into multiple classes. One task may be in charge of classifying a star into several classes (e.g., main sequence, dwarf, red giant, neutron, pulsar, etc.). Another task can focus on galaxy classification (e.g., spiral, barred spiral, elliptical, irregular, etc.). Rather than separating the problem into different tasks where each task is in charge of identifying one type of luminous object, one can combine the tasks together into a single parallel multitask problem where the hidden layer of a neural network shares patterns that are common to all classification tasks (see Fig. 2). The reasons explaining why learning often improves in accuracy and speed in this context is that training with many tasks in parallel on a single neural network induces information that accumulates in the training signals; if there exists properties common to several tasks, internal nodes can serve to represent common sub-concepts simultaneously.

Other Paradigms. Knowledge transfer can be performed using other learning and data analysis paradigms –mainly in the form of representational transfer– such as kernel methods, probabilistic methods, clustering, etc. (Raina et al. 2006; Evgeniou et al. 2005). For example, inductive transfer can take place in learning methods that assume a probabilistic distribution of the data by guaranteeing a form of relatedness among the distributions adopted across tasks (Raina et al. 2006). As an illustration, if learning to classify stars and galaxies both assume a mixture of normal densities to model the input-output or example-class distribution, one can force both distributions to have sets of parameters that are as similar as possible while preserving good generalization performance. In that case, shared knowledge can be interpreted as a set of assumptions about the data distribution for all tasks under analysis. The concept of knowledge transfer is also related to the problem of introducing new intermediate concepts during rule induction. In the inductive logic programming (ILP) setting, this is referred to as *predicate invention* (Stahl 1995).

Meta-Searching for Problem Solvers. A different research direction in inductive transfer explores complex scenarios where the software architecture itself evolves with experience (Schmidhuber 1997). The main idea is to divide a program into different components that can be reused during different stages of the learning process. As an illustration, one can work within the space of (self-delimiting binary) programs to propose an optimal ordered problem solver. The goal is to solve a sequence of problems, deriving one solution after the other, as optimally as possible. Ideally the system should be capable of exploiting previous solutions and of incorporating them into the solution to the current problem. This can be done by allocating computing time to the search for previous solutions that, if useful, become transformed into building blocks. We assume the current problem can be solved by copying or invoking previous pieces of code (i.e., building blocks or knowledge). In that case the mechanism will accept those solutions with substantial savings in computational time.

Inductive Transfer, Fig. 2 Example of multitask learning (functional transfer) applied to astronomical images



Domain Adaptation. A recent research direction in representational transfer seeks to adjust the model obtained in a source domain to account for differences exhibited in a new target domain. Unlike traditional studies in classification where both training and testing sets are assumed as realizations of the same joint input-output distribution, this *domain adaptation* approach either weakens or completely disregards such assumption (Ben-David et al. 2007, Daumé, et al. 2006, Storkey 2009). In addition, domain adaptation commonly assumes an abundance of labeled examples in the source domain, but little or no class labels in the target domain.

An example of these concepts lies in light curve classification from star samples obtained from different galaxies. A classification task set to differentiate different types of stars in a nearby source galaxy –where class labels are available– will experience a change in distribution as it moves to a target galaxy lying farther away –where class labels are unavailable. A major reason for such change is that at greater distances, less luminous stars fall below the detection threshold and more luminous stars are preferentially detected. The corresponding dataset shift (Quinonero-Candela et al. 2009) precludes the direct utilization of one single model across galaxies; it calls for a form of model adaptation to compensate for the change in the data distribution.

Domain adaptation has gained much attention recently, mainly due to the pervasive character of problems where distributions change over time. It assumes that the learning task remains constant, but the marginal and class posterior distributions between source and target domain may differ (as opposed to traditional transfer learning where tasks can in addition exhibit different input representations, i.e., different input spaces). Domain adaptation has been attacked from different angles: by searching for a single representation that unifies both source and target domains (Glorot et al. 2011); by proving error bounds as a function of empirical error and the distance between source and target distributions (Ben-David et al. 2010), within a co-training framework where target vectors are incorporated into the source training set based on confidence (Chen et al. 2011), by re-weighting source instances (Mansour et al. 2009), by using regularization terms to learn models that perform well on both source and target domains (Daumé et al. 2010), and several others.

Theoretical Work

Several studies have provided a theoretical analysis of the case where a learner uses experience from previous tasks to learn a new task. This process is often referred to as meta-learning or meta-generalization. The aim is to understand the conditions under which a learning algorithm can provide good generalizations when embedded in an environment made of related tasks. Although the idea of knowledge transfer is normally made implicit in the analysis, it is clear that the meta-learner extracts and exploits knowledge on every task to perform well on future tasks. Theoretical studies fall within a Bayesian model and within a probably approximately correct (PAC) model. The idea is to find not only the right hypothesis in a hypothesis space (base learning), but in addition to find the right hypothesis space in a family of hypothesis spaces (meta-learning).

We briefly review the main ideas behind these studies (Baxter 2000). We begin by assuming that the learner is embedded in a set of related tasks that share certain commonalities. Going back to the problem where a learner is designed for recognition of astronomical objects, the idea is to classify objects (e.g., stars, galaxies, nebulae, and planets) extracted from images mapping certain region of the sky. One way to transfer learning experience from one astronomical center to another is by sharing a meta-learner that carries a bias toward recognition of astronomical objects. In traditional learning, we assume a probability distribution $\mathbf{p}$ that indicates which examples are more likely to be seen in such a task. Now we assume that there is a more general distribution $\mathbf{P}$ over the space of all possible distributions. In essence, the meta-distribution $\mathbf{P}$ indicates which tasks are more likely to be found within the sequence of tasks faced by the meta-learner (distribution $\mathbf{p}$ indicates which examples are more likely to be seen in one task). In our example, the meta-distribution $\mathbf{P}$ peaks over tasks corresponding to classification of astronomical objects. Given a family of hypothesis spaces $\{\mathbf{H}\}$, the goal of the meta-learner is to find a hypothesis space $\mathbf{H}^*$ that minimizes a functional risk corresponding to the expected loss of the best possible hypothesis in each hypothesis space. In practice, since we ignore the form of $\mathbf{P}$, we need to draw samples $\mathbf{T_1}, \mathbf{T_2}, \ldots, \mathbf{T_n}$ to infer how tasks are distributed in our environment. To summarize, in the transfer learning scenario, our input is made of samples $\mathbf{T} = \{\mathbf{T_i}\}$, where each sample $\mathbf{T_i}$ is composed of examples. The goal of the meta-learner is to output a hypothesis space with a learning bias that generates accurate models for a new task.

Future Directions

The research community faces several challenges on how to efficiently transfer knowledge across tasks. One challenge involves devising learning architectures with an explicit representation of knowledge about models and algorithms, i.e., meta-knowledge. Most systems that integrate knowledge transfer mechanisms make an implicit assumption about the type of knowledge being transferred. This is indeed possible when strong assumptions are made on the relationship between the source and target tasks. For example, most approaches to domain adaptation work under strong assumptions about the similarity between the source and target tasks, imposing similar class posterior distributions, marginal distributions, or both. Ideally we would like to track the evolution of the source task to the target task to be able to justify any assumptions about their differences.

From a global perspective, it seems clear that proper treatment of the inductive transfer problem requires more than just statistical or mathematical techniques. Inductive transfer can be embedded in a complex artificial intelligence system that incorporates important components such as knowledge representation, search, planning, reasoning, etc. Without the incorporation of artificial intelligence components, we are forced to work with a large hypothesis space and a set of stringent assumptions about the nature of the discrepancy between the source and target tasks.

Cross-References

▶ Metalearning

Recommended Reading

Baxter J (2000) A model of inductive learning bias. J Artif Intell Res 12:149–198

Ben-David S, Blitzer J, Crammer K, Pereira F (2007) Analysis of representations for domain adaptation. Adv Neural Inf Process Syst 19:137–144

Ben-David S, Blitzer J, Crammer K, Kulesza A, Pereira F, Wortman J (2010) A theory of learning from different domains. Mach Learn Spec Issue Learn Mult Sources 79:151–175

Brazdil P, Giraud-Carrier C, Soares C, Vilalta R (2009) Metalearning: applications to data mining. Springer, Berlin

Caruana R (1993) Multitask learning: a knowledge-based source of inductive bias. In: Proceedings of the 10th international conference on machine learning (ICML), Amherst, pp 41-48

Chen M, Weinberger KQ, Blitzer J (2011) Co-training for domain adaptation. In: Advances in neural information processing systems (NIPS), Granada

Dai W, Yang Q, Xue G, Yu Y (2007) Boosting for transfer learning. In: Proceedings of the 24th international conference on machine learning (ICML), Corvallis, pp 193–200

Daumé H, Marcu D (2006) Domain adaptation for statistical classifiers. J Mach Learn Res 26:102–126

Daumé H, Kumar A, Saha A (2010) Co-regularization based semi-supervised domain adaptation. In: Advances in neural information processing systems (NIPS), Whistler

Evgeniou T, Micchelli CA, Pontil M (2005) Learning multiple tasks with kernel methods. J Mach Learn Res 6:615–637

Glorot X, Bordes A, Bengio Y (2011) Domain adaptation for large-scale sentiment classification: a deep learning approach. In: Proceedings of the 28th international conference on machine learning (ICML), Bellevue, pp 513–520

Mansour Y, Mohri M, Rostamizadeh A (2009) Domain adaptation with multiple sources. In: Advances in neural information processing systems (NIPS), Whistler, pp 1041–1048

Mihalkova L, Huynh T, Mooney RJ (2007) Mapping and revising markov logic networks for transfer learning. In: Proceedings of the 22nd AAAI conference on artificial intelligence, Vancouver, pp 608–614

Oblinger D, Reid M, Brodie M, de Salvo Braz R (2002) Cross-training and its application to skill-mining. IBM Syst J 41(3):449–460

Pratt L, Thrun S (1997) Second special issue on inductive transfer. Mach Learn 28:4175

Quinonero-Candela J, Sugiyama M, Schwaighofer A, Lawrence ND (2009) Dataset shift in machine learning. MIT Press, Cambridge

Raina R, Ng AY, Koller D (2006) Constructing informative priors using transfer learning. In: Proceedings of the 23rd international conference on machine learning (ICML), Pittsburgh, pp 713–720

Reid M (2004) Improving rule evaluation using multitask learning. In: Proceedings of the 14th international conference on ILP, Porto, pp 252–269

Schmidhuber J (1997) Shifting inductive bias with success-story algorithm, adaptive levin search, and incremental self-improvement. Mach Learn 28: 105–130

Stahl I (1995) Predicate invention in inductive logic programming. In: De Raedt L (ed) Advances in inductive logic programming. IOS Press, Amsterdam/Washington, DC, pp 34–47

Storkey A (2009) When training and test sets are different. In: Quinonero-Candela J, Sugiyama M, Schwaighofer A, Lawrence ND (eds) Dataset shift in machine learning. MIT Press, Cambridge, pp 3–28

Inequalities

▶ Generalization Bounds

Information Retrieval

Information retrieval (IR) is a set of techniques that extract from a collection of documents those that are relevant to a given query. Initially addressing the needs of librarians and specialists, the field has evolved dramatically with the advent of the World Wide Web. It is more general than *data retrieval*, whose purpose is to determine which documents contain occurrences of the keywords that make up a query. Whereas the syntax and semantics of data retrieval frameworks is strictly defined, with queries expressed in a totally formalized language, words from a natural language given no or limited structure are the medium of communication for information retrieval frameworks. A crucial task for an IR system is to index the collection of documents to make their contents efficiently accessible. The documents retrieved by the system are usually ranked by expected relevance, and the user who examines some of them might be able to provide feedback so that the query can be reformulated and the results improved.

In-Sample Evaluation

Synonyms

Within-sample evaluation

Definition

In-sample evaluation is an approach to ▶ algorithm evaluation whereby the learned model is evaluated on the data from which it was learned. This provides a biased estimate of learning performance, in contrast to ▶ holdout evaluation.

Cross-References

▶ Algorithm Evaluation

Instance

Synonyms

Case; Example; Item; Object

Definition

An *instance* is an individual object from the universe of discourse. Most learners create a model by analyzing a ▶ training set of instances. Most machine learning models take the form of a function from an ▶ instance space to an output space. In ▶ attribute-value learning, each instance is often represented as a vector of ▶ attribute values, each position in the vector corresponding to a unique attribute.

Instance Language

▶ Observation Language

Instance Space

Synonyms

Example space; Item space; Object space

Definition

An *instance space* is the space of all possible ▶ instances for some learning task. In ▶ attribute-value learning, the instance space is often depicted as a geometric space, one dimension corresponding to each attribute.

Instance-Based Learning

Eamonn Keogh
University of California-Riverside, Riverside, CA, USA

Synonyms

Analogical reasoning; Case-based learning; Memory-based; Nearest neighbor methods; Non-parametric methods

Definition

Instance-based learning refers to a family of techniques for ▶ classification and ▶ regression, which produce a class label/predication based on the similarity of the query to its nearest neighbor(s) in the training set. In explicit contrast to other methods such as ▶ decision trees and ▶ neural networks, instance-based learning algorithms do not create an abstraction from specific instances. Rather, they simply store all the data, and at query time derive an answer from an examination of the query's ▶ nearest neighbor (s).

Somewhat more generally, instance-based learning can refer to a class of procedures for solving new problems based on the solutions of similar past problems.

Motivation and Background

Most instance-based learning algorithms can be specified by determining the following four items:

1. Distance measure: Since the notion of similarity is being used to produce class label/prediction, we must explicitly state what similarity/distance measure to use. For real-valued data, Euclidean distance is a popular choice and may be optimal under some assumptions.
2. Number of neighbors to consider: It is possible to consider any number from one to all neighbors. This number is typically denoted as k.
3. Weighting function: It is possible to give each neighbor equal weight, or to weight them based on their distance to the query.
4. Mapping from local points: Finally, some method must be specified to use the (possibly weighted) neighbors to produce an answer. For example, for regression the output can be the weighted mean of the k nearest neighbors, or for classification the output can be the majority vote of the k nearest neighbors (with some specified tie-breaking procedure).

Since instance-based learning algorithms defer all the work until a query is submitted, they are sometimes called lazy algorithms (in contrast to eager learning algorithms, such as decision trees). Beyond the setting of parameters/distance measures/mapping noted above, one of the main research issues with instance-based learning algorithms is mitigating their expensive classification time, since a naïve algorithm would require comparing the distance for the query to every point in the database. Two obvious solutions are indexing the data to achieve a sublinear search, and numerosity reduction (data editing) (Wilson and Martinez 2000).

Further Reading

The best distance measure to use with an instance-based learning algorithms is the subject of active research. For the special case of time series data alone, there are at least one hundred methods (Ding et al. 2008). Conferences such as ICML, SIGKDD, etc. typically have several papers each year which introduce new distance measures and/or efficient search techniques.

Recommended Reading

Aha DW, Kibler D, Albert MK (1991) Instance-based learning algorithms. Mach Learn 6:37–66

Ding H, Trajcevski G, Scheuermann P, Wang X, Keogh EJ (2008) Querying and mining of time series data: experimental comparison of representations and distance measures. PVLDB 1(2):1542–1552

Wilson DR, Martinez TR (2000) Reduction techniques for exemplar-based learning algorithms. Mach Learn 38(3):257–286

Instance-Based Reinforcement Learning

William D. Smart
Washington University in St. Louis, St. Louis, MO, USA

Synonyms

Kernel-based reinforcement learning

Definition

Traditional reinforcement-learning (RL) algorithms operate on domains with discrete state spaces. They typically represent the value function in a table, indexed by states, or by state–action pairs. However, when applying RL to domains with continuous state, a tabular representation is no longer possible. In these cases, a common approach is to represent the value function by storing the values of a small set of states (or state–action pairs), and interpolating these values to other, unstored, states (or state–action pairs). This approach is known as instance-based reinforcement learning (IBRL). The instances are the explicitly stored values, and the interpolation is typically done using well-known instance-based supervised learning algorithms.

Motivation and Background

Instance-Based Reinforcement Learning (IBRL) is one of a set of value-function approximation techniques that allow standard RL algorithms to deal with problems that have continuous state spaces. Essentially, the tabular representation of the value function is replaced by an instance-based supervised learning algorithm and the rest of the RL algorithm remains unaltered. Instance-based methods are appealing because each stored instance can be viewed as analogous to one cell in the tabular representation. The interpolation method of the instance-based learning algorithm then blends the value between these instances.

IBRL allows generalization of value across the state (or state–action) space. Unlike tabular representations it is capable of returning a value approximation for states (or state–action pairs) that have never been directly experienced by the system. This means that, in theory, fewer experiences are needed to learn a good approximation to the value function and, hence, a good control policy. IBRL also provides a more compact representation of the value function than a table does. This is especially important in problems with multi-dimensional continuous state spaces. A straightforward discretization of such a space results in an exponential number of table cells. This, in turn, leads to an exponential increase in the amount of training experiences needed to obtain a good approximation of the value function.

An additional benefit of IBRL over other value-function approximation techniques, such as artificial neural networks, is the ability to bound the predicted value of the approximation. This is important, since it allow us to retain some of the theoretical non-divergence results for tabular representations.

Structure of Learning System

IBRL can be used to approximate both the state value function and the state–action value function. For problems with discrete actions, it is common to store a separate value function for each action. For continuous actions, the (continuous) state and action vectors are often concatenated, and VFA is done over this combined domain. For clarity, we will discuss only the state value function here, although our comments apply equally well to the state–action value function.

The Basic Approach

IBRL uses an instance-based supervised learning algorithm to replace the tabular value function representation of common RL algorithms. It maintains a set of states, often called basis points, and their associated values, using them to provide a value-function approximation for the entire state space. These exemplar states can be obtained in a variety of ways, depending on the nature of the problem. The simplest approach is to sample, either regularly or randomly, from the state space. However, this approach can result in an unacceptably large number of instances, especially if the state space is large, or has high dimension. A better approach is to use states encountered by the learning agent as it follows trajectories in the state space. This allows the representational power of the approximation algorithm to be focused on areas of the space in which the learning agent is likely to be. This, too, can result in a large number of states, if the agent is long-lived. A final approach combines the previous two by sub-sampling from the observed states.

Each stored instance state has a value associated with it, and an instance-based supervised learning algorithm is used to calculate the value of all other states. While any instance-based algorithm can be used, kernel-based algorithms have proven to be popular. Algorithms such as locally weighted regression (Smart and Kaelbling 2000), and radial basis function networks (Kretchmar and Anderson 1997) are commonly seen in the literature. These algorithms make some implicit assumptions about the form of the value function and the underlying state space, which we discuss below. For a state s, the kernel-based value-function approximation $V(s)$ is

$$V(s) = \frac{1}{\eta} \sum_{i=1}^{n} \phi(s, s_i) V(s_i), \tag{1}$$

where the s_i values are the n stored basis points, η is a normalizer,

$$\eta = \sum_{i=1}^{n} \phi(s, s_i), \tag{2}$$

and ϕ is the kernel function. A common choice for ϕ is an exponential kernel,

$$\phi(s, t) = e^{\frac{(s-t)^2}{\sigma^2}}, \tag{3}$$

where σ is the kernel bandwidth. The use of kernel-based approximation algorithms is well motivated, since they respect Gordon's non-divergence conditions (Gordon 1995), and also Szepesvári and Smart's convergence criteria (Szepesvári and Smart 2004).

As the agent gathers experience, the value approximations at each of the stored states and, optionally, the location and bandwidth of the states must be updated. Several techniques, often based on the temporal difference error, have been proposed, but the problem remains open. An alternative to on-line updates is a batch approach, which relies on storing the experiences generated by the RL agent, composing these into a discrete MDP, solving this MDP exactly, and then using supervised learning techniques on the states and their associated values. This approach is known as fitted value iteration (Szepesvári and Munos 2005).

Examples of IBRL Algorithms

Several IBRL algorithms have been reported in the literature. Kretchmar and Anderson (1997) presented one of the first IBRL algorithms. They used a radial basis function (RBF) network to approximate the state–action value function for the well-known mountain-car test domain. The temporal difference error of the value update is used to modify the weights, centers, and variances of the RBF units, although they noted that it was not particularly effective in producing good control policies.

Smart and Kaelbling (2000) used locally weighted learning algorithms and a set of heuristic rules to approximate the state–action value function. A set of states, sampled from those experienced by the learning agent, were stored along with their associated values. One approximation was stored for each discrete action. Interpolation between these exemplars was done by locally weighted averaging or locally weighted regression, supplemented with heuristics to avoid extrapolation and over-estimation. Learning was done on-line, with new instances being added as the learning agent explored the state space. The algorithm was shown to be effective in practice, but offered no theoretical guarantees.

Ormoneit and Sen (2002) presented an offline kernel-based reinforcement-learning algorithm that stores experiences (s_i, a_i, r_i, s_i') as the instances, and uses these to approximate the state–action value function for problems with discrete actions. For a given state s and action a, the state–action value $Q(s, a)$ is approximated as

$$\hat{Q}(s, a) = \frac{1}{\eta_{s,a}} \sum_{i | a_i = a} \phi\left(\frac{d(s, s_i)}{\sigma}\right) \left[r_i + \gamma \max_{a'} \hat{Q}(s_i', a')\right], \tag{4}$$

where ϕ is a kernel function, σ is the kernel bandwidth, γ is the RL discount factor, and $\eta_{s,a}$ is a normalizing term,

$$\eta_{s,a} = \sum_{i|a_i=a} \phi\left(\frac{d(s, s_i)}{\sigma}\right). \qquad (5)$$

They showed that, with enough basis points, this approximation converges to the true value function, under some reasonable assumptions. However, they provide no bound on the number of basis points needed to provide a good approximation to the value function.

Assumptions

IBRL makes a number of assumptions about the form of the value function, and the underlying state space. The main assumptions are that state similarity is well measure by (weighted) Euclidean distance. This implicity assumes that the underlying state space be metric, and is a topological disk. Essentially, this means that stattes that are close to each other in the state space have similar value. This is clearly not true for states between which the agent cannot move, such as those on the opposite sides of a thin wall. In this case, there is a discontinuity in the state space, introduced by the wall, which is not well modeled by the instance-based algorithm.

Instance-based function approximation algorithms assume that the function they model is smooth and continuous between the basis points. Any discontinuities in the function tend to get "smoothed out" in the approximation. This assumption is especially problematic for value-function approximation, since it allows value on one side of the discontinuity to affect the approximation on the other. If the location of the discontinuity is known, and we are able to allocate an arbitrary number of basis points, we can overcome this problem. However, in practical applications of RL, neither of these is feasible, and the problem of approximating the value function at or near discontinuities remains an open one.

Problems and Drawbacks

Although IBRL has been shown to be effective on a number of problems, it does have a number of drawbacks that remain unaddressed. Instance-based approximation algorithms are often expensive in terms of storage, especially for long-lived agents. Although the literature contains many techniques for editing the basis set of instance-based approximators, these techniques are generally for a supervised learning setting, where the utility of a particular edit can be easily evaluated. In the RL setting, we lack the ground truth available to supervised learning, making the evaluation of edits considerably more difficult. Additionally, as the number of basis points increases, so does the time needed to perform an approximation. This limitation is significant in the RL setting, since many such value predictions are needed on every step of the accompanying RL algorithm.

The value of a particular state, s, is calculated by blending the values from other nearby states, s_i. This is problematic if it is not possible to move from state s to each of the states s_i. The value of s should only be influenced by the value of states reachable from s, but this condition is not enforced by standard instance-based approximation algorithms. This leads to problems when modeling discontinuities in the value function, as noted above, and in situations where the system dynamics constrain the agent's motion, as in the case of a "one-way door" in the state space.

IBRL also suffers badly from the curse of dimen-sionality; the number of points needed to adequately represent the value function is exponential in the dimensionality of the state space. However, by using only states actually experienced by the learning agent, we can lessen the impact of this problem. By using only observed states, we are explicitly modeling the manifold over which the system state moves. This manifold is embedded in the full state space and, for many real-world problems, has a lower dimensionality than the full space. The Euclidean distance metric used by many instance-based algorithms will not accurately measure distance along this manifold. In practice, the manifold over which the system state moves will be locally Euclidean for problems with smooth, continuous dynamics. As a result, the assumptions of instance-based function approximators are valid locally and the approximations are of reasonable quality.

Cross-References

- ▶ Curse of Dimensionality
- ▶ Instance-Based Learning
- ▶ Locally Weighted Learning
- ▶ Reinforcement Learning
- ▶ Value Function Approximation

Recommended Reading

Gordon GJ (1995) Stable function approximation in dynamic programming. In: Proceedings of the twelfth international conference on machine learning, Tahoe City, pp 261–268

Kretchmar RM, Anderson CW (1997) Comparison of CMACs and radial basis functions for local function approximators in reinforcement learning. In: International conference on neural networks, Houston, vol 2, pp 834–837

Ormoneit D, Sen Ś (2002) Kernel-based reinforcement learning. Mach Learn 49(2–3):161–178

Smart WD, Kaelbling LP (2000) Practical reinforcement learning in continuous spaces. In: Proceedings of the seventeenth international conference on machine learning (ICML 2000), Stanford, pp 903–910

Szepesvári C, Munos R (2005) Finite time bounds for sampling based fitted value iteration. In: Proceedings of the twenty-second international conference on machine learning (ICML 2005), Bonn, pp 880–887

Szepesvári C, Smart WD (2004) Interpolation-based Q-learning. In: Proceedings of the twenty-first international conference on machine learning (ICML 2004), Banff, pp 791–798

Intelligent Backtracking

Synonyms

Dependency directed backtracking

Definition

Intelligent backtracking is a general class of techniques used to enhance search and constraint satisfaction algorithms. Backtracking is a general mechanism in search where a problem solver encounters an unsolvable search state and backtracks to a previous search state that might be solvable. Intelligent backtracking mechanisms provide various ways of selecting the backtracking point based on past experience in a way that is likely to be fruitful.

Intent Recognition

▶ Inverse Reinforcement Learning

Internal Model Control

Synonyms

Certainty equivalence principle; Model-based control

Definition

Many advanced controllers for nonlinear systems require knowledge of the model of the dynamics of the system to be controlled. The system dynamics is often called an "internal model," and the resulting controller is model-based. If the model is not known, it can be learned with function approximation techniques. The learned model is subsequently used as if it were correct in order to synthesize a controller – the control literature calls this assumption the "certainty equivalence principle."

Interval Scale

An **interval** measurement scale ranks the data, and the differences between units of measure can be calculated by arithmetic. However, *zero* in the interval level of measurement means neither "nil" nor "nothing" as *zero* in arithmetic means. See ▶ Measurement Scales.

Inverse Entailment

Definition

Inverse entailment is a ▶ generality relation in ▶ inductive logic programming. More specifically, when learning from entailment using a background theory B, a hypothesis H covers an example e, relative to the background theory B if and only if $B \wedge H \models e$, that is, the background theory B and the hypothesis H together entail the example (see ▶ entailment). For instance, consider the background theory B:

```
bird :- blackbird.
bird :- ostrich.
```

and the hypothesis H:

```
flies :- bird.
```

Together $B \wedge H$ entail the example e:

```
flies :- blackbird, normal.
```

This can be decided through deductive inference. Now when learning from entailment in inductive logic programming, one starts from the example e and the background theory B, and the aim is to induce a rule H that together with B entails the example. Inverting entailment is based on the observation that $B \wedge H \models e$ is logically equivalent to $B \wedge \neg e \models \neg H$, which in turn can be used to compute a hypothesis H that will cover the example relative to the background theory. Indeed, the negation of the example is $\neg e$:

```
blackbird.
normal.
:-flies.
```

and together with B this entails $\neg H$:

```
bird.
:-flies.
```

The principle of inverse entailment is typically employed to compute the ▶ bottom clause, which is the most specific clause covering the example under entailment. It can be computed by generating the set of all facts (true and false) that are entailed by $B \wedge \neg e$ and negating the resulting formula $\neg H$.

Cross-References

▶ Bottom Clause
▶ Entailment
▶ Inductive Logic Programming
▶ Logic of Generality

Inverse Optimal Control

▶ Inverse Reinforcement Learning

Inverse Reinforcement Learning

Pieter Abbeel[1] and Andrew Y. Ng[2]
[1]EECS Department, UC Berkeley, Stanford, CA, USA
[2]Computer Science Department, Stanford University, Stanford, CA, USA
[3]Stanford University, Stanford, CA, USA

Synonyms

Intent recognition; Inverse optimal control; Plan recognition

Definition

Inverse reinforcement learning (inverse RL) considers the problem of extracting a reward function from observed (nearly) optimal behavior of an expert acting in an environment.

Motivation and Background

The motivation for inverse RL is twofold:

- For many RL applications, it is difficult to write down an explicit reward function specifying how different desiderata should be traded off exactly. In fact, engineers often spend significant effort tweaking the reward function such that the optimal policy corresponds to performing the task they have in mind. For example, consider the task of driving a car well. Various desiderata have to be traded off, such as speed, following distance, lane preference, frequency of lane changes, distance from the curb, etc. Specifying the reward function for the task of driving requires explicitly writing down the trade-off between these features.

Inverse RL algorithms provide an efficient solution to this problem in the apprenticeship learning setting – when an expert is available to demonstrate the task. Inverse RL algorithms exploit the fact that an expert demonstration implicitly encodes the reward function of the task at hand.

- Reinforcement learning and related frameworks are often used as computational models for animal and human learning (Watkins 1989; Schmajuk and Zanutto 1997; Touretzky and Saksida 1997). Such models are supported both by behavioral studies and by neurophysiological evidence that reinforcement learning occurs in bee foraging (Montague et al. 1995) and in songbird vocalization (Doya and Sejnowski 1995). It seems clear that in examining animal and human behavior, we must consider the reward function as an unknown to be ascertained through empirical investigation, particularly when dealing with multiatttribute reward functions. Consider, for example, that the bee might weigh nectar ingestion against flight distance, time, and risk from wind and predators. It is hard to see how one could determine the relative weights of these terms a priori. Similar considerations apply to human economic behavior, for example. Hence, inverse reinforcement learning is a fundamental problem of theoretical biology, econometrics, and other scientific disciplines that deal with reward-driven behavior.

Structure of the Learning System

Preliminaries and Notation

A Markov decision process (MDP) is a tuple $\langle S, A, T, \gamma, D, R\rangle$, where S is a finite set of states, A is a set of actions, $T = \{P_{sa}\}$ is a set of state-transition probabilities (here, P_{sa} is the state transition distribution upon taking action a in state s), $\gamma \in [0, 1)$ is a discount factor, D is the distribution over states for time zero, and $R : S \mapsto \mathbb{R}$ is the reward function.

A policy π is a mapping from states to probability distributions over actions. We let Π denote the set of all stationary policies (We restrict attention to stationary policies, since it is well known that there exists a stationary policy that is optimal for infinite horizon MDPs.). The value of a policy π is given by

$$V(\pi) = \mathrm{e}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi\right].$$

The expectation is taken with respect to the random state sequence $s_0, s_1, s_2, \ldots$ drawn by starting from a state $s_0 \sim D$ and picking actions according to π.

Let $\mu_S(\pi)$ be the discounted distribution over states when acting according to the policy π. In particular, for a discrete state space, we have that $[\mu_S(\pi)](s) = \sum_{t=0}^{\infty} \gamma^t \mathrm{Prob}(s_t = s|\pi)$. (In the case of a continuous state space, we replace $\mathrm{Prob}(s_t = s|\pi)$ by the appropriate probability density function.) Then, we have that

$$V(\pi) = R^\top \mu_S(\pi).$$

Thus, the value of a policy π when starting from a state s_0 is linear in the reward function.

Often the reward function R can be represented more compactly. Let $\phi : S \to \mathbb{R}^n$ be a feature map. A typical assumption in inverse RL is to assume the reward function R is a linear combination of the features ϕ: $R(s) = w^\top \phi(s)$. Then, we have that the value of a policy π is linear in the reward function weights w:

$$\begin{aligned} V(\pi) &= E[\textstyle\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi] \\ &= E[\textstyle\sum_{t=0}^{\infty} \gamma^t w^\top \phi(s_t)|\pi] \\ &= w^\top E[\textstyle\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi] \\ &= w^\top \mu_\phi(\pi). \end{aligned} \quad (1)$$

Here, we used linearity of expectation to bring w outside of the expectation. The last equality defines the vector of *feature expectations* $\mu_\phi(\pi) = E\big[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi\big]$.

We assume access to demonstrations by some expert. We denote the expert's policy by π^*. Specifically, we assume the ability to observe trajectories (state sequences) generated by the expert starting from $s_0 \sim D$ and taking actions according to π^*.

Characterization of the Inverse RL Solution Set

A reward function R is consistent with the policy π^* being optimal if and only if the value obtained when acting according to the policy π^* is at least as high as the value obtained when acting according to any other policy π, or equivalently:

$$U(\pi^*) \geq U(\pi) \ \ \forall \pi \in \Pi. \tag{2}$$

Using the fact that $U(\pi) = R^\top \mu_S(\pi)$, we can equivalently write the conditions of Eq. (2) as a set of linear constraints on the reward function R:

$$R^\top \mu_S(\pi^*) \geq R^\top \mu_S(\pi) \ \ \forall \pi \in \Pi. \tag{3}$$

The state distribution $\mu_S(\pi)$ does not depend on the reward function R. Thus, Eq. (3) is a set of linear constraints in the reward function, and we can use a linear program (LP) solver to find a reward function consistent with the policy π^* being optimal. Strictly speaking, Eq. (3) solves the inverse RL problem. However, to apply inverse RL in practice, the following three issues need to be addressed:

1. **Reward function ambiguity.** Typically a large set of reward functions satisfy all the constraints of Eq. (3). One such a reward function that satisfies all the constraints for any MDP is the all-zeros reward function (it is consistent with any policy being optimal). Clearly, the all-zeros reward function is not a desirable answer to the inverse RL problem. More generally, this observation suggests not all reward functions satisfying Eq. (3) are of equal interest and raises the question of how to recover reward functions that are of interest to the inverse RL problem.
2. **Statistical efficiency.** Often the state space is very large (or even infinite), and we do not have sufficiently many expert demonstrations available to accurately estimate $\mu(\cdot\,; \pi^*)$ from data.
3. **Computational efficiency.** The number of constraints in Eq. (3) is equal to the number of stationary policies $|\Pi|$ and grows quickly with the number of states and actions of the MDP. For finite-state-action MDPs, we have $|A|^{|S|}$ constraints. So, even for small state and action spaces, feeding all the constraints of Eq. (3) into an LP solver becomes quickly impractical. For continuous state-action spaces, the formulation of Eq. (3) has an infinite number of constraints, and thus using a standard LP solver to find a feasible reward function R is impossible.

In the following sections, we address these three issues.

Reward Function Ambiguity

As observed above, typically a large set of reward functions satisfy all the constraints of Eq. (3). To obtain a single reward function, it is natural to reformulate the inverse RL problem as an optimization problem. We describe one standard approach for disambiguation. Of course many other formulations as an optimization problem are possible.

Similar to common practice in support vector machines research, one can maximize the (soft) margin by which the policy π^* outperforms all other policies. As is common in structured prediction tasks ((see, e.g., Taskar et al. (2003)), one can require the margin by which the policy π^* outperforms another policy π to be larger when π differs more from π^*, as measured according to some function $h(\pi^*, \pi)$. The resulting formulation (Ratliff et al. 2006) is

$$\begin{aligned} \min_{R,\xi} \quad & \|R\|_2^2 + C\xi \\ \text{s.t. } & R^\top \mu_S(\pi^*) \geq R^\top \mu_S(\pi) + h(\pi^*, \pi) - \xi \\ & \forall \pi \in \Pi. \end{aligned} \tag{4}$$

For the resulting optimal reward function to correspond to a desirable solution to the inverse RL problem, it is important that the objective and the margin scaling encode the proper prior knowledge. If a sparse reward function is suggested by prior knowledge, then a 1-norm might be more appropriate in the objective. An example of a margin scaling function for a discrete MDP is the number of states in which the action prescribed by the policy π differs from the action prescribed by the expert policy π^*. If the expert has only been observed in a small number of states, then one could restrict attention to these states when evaluating this margin scaling function.

Another way of encoding prior knowledge is by restricting the reward function to belong to a certain functional class, for example, the set of functions linear in a specified set of features. This approach is very common and is also important for statistical efficiency. It will be explained in the next section.

Remark When using inverse RL to help us specify a reward function for a given task based on an expert demonstration, it is not necessary to explicitly resolve the ambiguities in the reward function. In particular, one can probably perform as well as the expert without matching the expert's reward function. More details are given in Sect. "A Generative Approach to Inverse RL".

Statistical Efficiency

As formulated thus far, solving the inverse RL problem requires the knowledge (or accurate statistical estimates) of $\mu_S(\pi^*)$. For most practical problems, the number of states is large (or even infinite), and thus accurately estimating $\mu_S(\pi^*)$ requires a very large number of expert demonstrations. This (statistical) problem can be resolved by restricting the reward function to belong to a prespecified class of functions. The common approach is to assume the reward function R can be expressed as a linear combination of a known set of features. In particular, we have $R(s) = w^\top \phi(s)$. Using this assumption, we can use the expression for the value of the policy π from Eq. (1).

Rewriting Eq. (4), we now have the following constraints in the reward weights w:

$$\begin{aligned} \min_{w,\xi} \quad & \|w\|_2^2 + C\xi \\ \text{s.t. } & w^\top \mu_\phi(\pi^*) \geq w^\top \mu_\phi(\pi) + h(\pi^*, \pi) - \xi \\ & \forall \pi \in \Pi. \end{aligned} \tag{5}$$

This new formulation only requires estimates of the expected feature counts $\mu_\phi(\pi^*)$, rather than estimates of the distribution over the state space $\mu_S(\pi^*)$. Assuming the number of features is smaller than the number of states, this significantly reduces the number of expert demonstrations required.

Computational Efficiency

For concreteness, we will consider the formulation of Eq. (6). Although the number of variables is only equal to the number of features in the reward function, the number of constraints is very large (equal to the number of stationary policies). As a consequence, feeding the problem into a standard quadratic programming (QP) solver will not work.

Ratliff et al. (2006) suggested a formal computational approach to solving the inverse RL problem, using standard techniques from convex optimization, which provide convergence guarantees. More specifically, they used a subgradient method to optimize the following equivalent problem:

$$\begin{aligned} \min_{w,\xi} \|w\|_2^2 + C \max_{\pi \in \Pi} \Big(& w^\top \mu_\phi(\pi) + h(\pi^*, \pi) \\ & - w^\top \mu_\phi(\pi^*) \Big) . \end{aligned} \tag{6}$$

In each iteration, to compute the subgradient, it is sufficient to find the optimal policy with respect to a reward function that is easily determined from the current reward weights w and the margin scaling function $h(\pi^*, \cdot)$. In more recent work, Ratliff et al. (2007) proposed a boosting algorithm to solve a formulation similar to Eq. (6), which also includes feature selection.

A Generative Approach to Inverse RL

Abbeel and Ng (2004) made the following observation, which resolves the ambiguity problem in a completely different way: if, for a policy π, we have that $\mu_\phi(\pi) = \mu_\phi(\pi^*)$, then the following holds:

$$U(\pi) = w^\top \mu_\phi(\pi) = w^\top \mu_\phi(\pi^*) = U(\pi^*),$$

no matter what the value of w is. Thus, to perform as well as the expert, it is sufficient to find a policy that attains the same expected feature counts μ_ϕ as the expert.

Abbeel and Ng provide an algorithm that finds a policy π satisfying $\mu_\phi(\pi) = \mu_\phi(\pi^*)$. The algorithm iterates over two steps: (i) Generate a reward function by solving a QP. (ii) Solve the MDP for the current reward function.

In contrast to the previously described inverse RL methods, which focus on merely recovering a reward function that could explain the expert's behavior, this inverse RL algorithm is shown to find a policy that performs at least as well as the expert. The algorithm is shown to converge in a polynomial number of iterations.

Apprenticeship Learning: Inverse RL Versus Imitation Learning

Inverse RL alleviates the need to specify a reward function for a given task when expert demonstrations are available. Alternatively, one could directly estimate the policy of the expert using standard a machine-learning algorithm, since it is simply a mapping from state to action. The latter approach, often referred to as imitation learning or behavioral cloning (links), has been successfully tested on a variety of tasks, including learning to fly in a fixed-wing flight simulator (Sammut et al. 1992) and learning to drive a car (Pomerleau 1989; Abbeel and Ng 2004).

The imitation learning approach can be expected to be successful whenever the policy class to be considered can be learned efficiently from data. In contrast, the inverse RL approach relies on having a reward function that can be estimated efficiently from data.

Cross-References

- ▶ Apprenticeship Learning
- ▶ Reinforcement Learning
- ▶ Reward Shaping

Recommended Reading

Abbeel P, Ng AY (2004) Apprenticeship learning via inverse reinforcement learning. In: Proceedings of ICML, Alberta

Doya K, Sejnowski T (1995) A novel reinforcement model of birdsong vocalization learning. Neural Inf Process Syst 7:101

Montague PR, Dayan P, Person C, Sejnowski TJ (1995) Bee foragin in uncertain environments using predictive hebbian learning. Nature 377(6551):725–728

Pomerleau D (1989) Alvinn: an autonomous land vehicle in a neural network. In: NIPS 1, Denver

Ratliff N, Bagnell J, Zinkevich M (2006) Maximum margin planning. In: Proceedings of ICML, Pittsburgh

Ratliff N, Bradley D, Bagnell J, Chestnutt J (2007) Boosting structured prediction for imitation learning. Neural Inf Process Syst 19:1153–1160

Sammut C, Hurst S, Kedzier D, Michie D (1992) Learning to fly. In: Proceedings of ICML, Aberdeen

Schmajuk NA, Zanutto BS (1997) Escape, avoidance, and imitation. Adapt Behav 6:63–129

Taskar B, Guestrin C, Koller D (2003) Max-margin markov networks. In: Neural information processing systems conference (NIPS03), Vancouver

Touretzky DS, Saksida LM (1997) Operant conditioning in skinnerbots. Adapt Behav 5:219–47

Watkins CJ (1989) Models of delayed reinforcement learning. Ph.D. thesis, Psychology Department, Cambridge University

Inverse Resolution

Definition

Inverse resolution is, as the name indicates, a rule that inverts resolution. This follows the idea of induction as the inverse of deduction formulated in the ▶ logic of generality. The resolution rule is the best-known deductive inference rule, used in many theorem provers and logic programming systems. ▶ Resolution starts from two ▶ clauses and derives the resolvent, a clause that is entailed by the two clauses. This can be graphically represented using the following schema (for propositional logic).

$$\frac{h \leftarrow g, a_1, \ldots, a_n \text{ and } g \leftarrow b_1, \ldots, b_m}{h \leftarrow b_1, \ldots, b_m, a_1, \ldots, a_n}.$$

Inverse resolution operators, such as *absorption* (17) and *identification* (17), invert this process.

To this aim, they typically assume the resolvent is given together with *one* of the original clauses and then derive the missing clause. This leads to the following two operators, which start from the clauses below and induce the clause above the line.

$$\frac{h \leftarrow g, a_1, \ldots, a_n \text{ and } g \leftarrow b_1, \ldots, b_m}{h \leftarrow b_1, \ldots, b_m, a_1, \ldots, a_n \text{ and } g \leftarrow b_1, \ldots, b_m},$$

$$\frac{h \leftarrow g, a_1, \ldots, a_n \text{ and } g \leftarrow b_1, \ldots, b_m}{h \leftarrow b_1, \ldots, b_m, a_1, \ldots, a_n \text{ and } h \leftarrow g, a_1, \ldots, a_n}$$

The operators are shown here only for the propositional case, as the first order case is more involved as it requires one to deal with substitutions as well as inverse substitutions.

As one example, consider the clauses

1. flies :- bird, normal.
2. bird :- blackbird.
3. flies :- blackbird, normal.

Here, (3) is the resolvent of (1) and (2). Furthermore, starting from (3) and (2), the absorption operator would generate (1), and starting from (3) and (1), the identification operator would generate (2).

Cross-References

▶ First-Order Logic
▶ Logic of Generality

Is More General Than

▶ Logic of Generality

Is More Specific Than

▶ Logic of Generality

Isotonic Calibration

▶ Classifier Calibration

Item

▶ Instance

Item Space

▶ Instance Space

Iterative Algorithm

▶ *K*-Medoids Clustering

Iterative Classification

▶ Collective Classification

Iterative Computation

▶ *K*-Means Clustering

J

Junk Email Filtering

► Text Mining for Spam Filtering

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

K

k-Armed Bandit

Shie Mannor
Israel Institute of Technology, Haifa, Israel

Synonyms

Multi-armed bandit; Multi-armed bandit problem

Definition

In the classical k-armed bandit problem, there are k alternative arms, each with a stochastic reward whose probability distribution is initially unknown. A decision maker can try these arms in some order, which may depend on the rewards that have been observed so far. A common objective in this context is to find a policy for choosing the next arm to be tried, under which the sum of the expected rewards comes as close as possible to the ideal reward, that is, the expected reward that would be obtained if it were to try the "best" arm at all times. There are many variants of the k-armed bandit problem that are distinguished by the objective of the decision maker, the process governing the reward of each arm, and the information available to the decision maker at the end of every trial.

Motivation and Background

k-Armed bandit problems are a family of sequential decision problems that are among the most studied problems in statistics, control, decision theory, and machine learning. In spite of their simplicity, they encompass many of the basic problems of sequential decision making in uncertain environments such as the tradeoff between exploration and exploitation.

There are many variants of bandit problems including Bayesian, Markovian, adversarial, budgeted, and exploratory variants. Bandit formulations arise naturally in multiple fields and disciplines including communication networks, clinical trials, search theory, scheduling, supply chain automation, finance, control, information technology, etc. (Berry and Fristedt 1985; Cesa-Bianchi and Lugosi 2006; Gittins 1989).

The term "multi-armed bandit" is borrowed from the slang term for a slot machine (the one-armed bandit), where a decision maker has to decide whether to insert a coin into the gambling machine and pull a lever possibly getting a significant reward, or to quit without spending any money.

Theory

We briefly review some of the most popular bandit variants.

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

The Stochastic *k*-Armed Bandit Problem

The classical stochastic bandit problem is described as follows. There are k arms (or machines or actions) and a single decision maker (or controller or agent). Each arm corresponds to a discrete time Markov process. At each timestep, the decision maker observes the current state of each arm's process and selects one of the arms. As a result, the decision maker obtains a reward from the process of the selected arm and the state of the corresponding process changes. Arms that are not selected are "frozen" and their processes remain in the same state. The objective of the decision maker is to maximize her (discounted) reward.

More formally, let the state of arm n's process at stage t be $x_n(t)$. Then, if the decision maker selects arm $m(t)$ at time t we have that:

$$x_n(t+1) = \begin{cases} x_n(t) & n \neq m(t) \\ f_n(x_n(t), \omega) & n = m(t) \end{cases},$$

where $f_n(x, \omega)$ is a function that describes the (possibly stochastic) transition probability of the n-th process and accepts the state of the n-th process and a random disturbance ω.

The reward the decision maker receives at time t is a function of the current state and a random element: $r(x_{m(t)}(t), \omega)$. The objective of the decision maker is to maximize her cumulative discounted reward. That is, she wishes to maximize

$$V = \mathbf{E}^{\pi}\left[\sum_{t=1}^{\infty} \gamma^t r(x_{m(t)}(t), \omega_t)\right],$$

where $\mathbf{E}^{\pi}$ is the expectation obtained when following policy π and γ is a discount factor ($0 < \gamma < 1$). A policy is a decision rule for selected arms as a function of the state of the processes.

This problem can be solved using ▶ dynamic programming, but the state space of the joint Markov decision process is exponential in the number of arms. Moreover, the dynamic programming solution does not reveal the important structural properties of the solution.

Gittins and Jones (1972) showed that there exists an optimal index policy. That is, there is a function that maps the state of each arm to real number (the "index") such that the optimal policy is to choose the arm with the highest index at any given time. Therefore, the stochastic bandit problem reduces to the problem of computing the index, which can be easily done in many important cases.

Regret Minimization for the Stochastic *k*-Armed Bandit Problem

A different flavor of the bandit problem focuses on the notion of regret, or learning loss. In this formulation, there are k arms as before and when selecting arm m a reward that is independent and identically distributed is given (the reward depends only on the identity of the arm and not on some internal state or the results of previous trials). The decision maker's objective is to obtain high expected reward. Of course, if the decision maker had known the statistical properties of each arm, she would have always chosen the arm with the highest expected reward. However, the decision maker does not know the statistical properties of the arms in advance, in this setting.

More formally, if the reward when choosing arm m has expectation r_m, the regret is defined as:

$$r(t) = t \cdot \max_{1 \le m \le k} r_m - \mathbf{E}^{\pi}\left[\sum_{\tau=1}^{t} r(\tau)\right], \qquad (1)$$

where $r(t)$ is sampled from the arm $m(t)$. This quantity represents the expected loss for not choosing the arm with the highest expected reward on every timestep.

This variant of the bandit problem highlights the tension between acquiring information (exploration) and using the available information (exploitation). The decision maker should carefully balance between the two since if she chooses to only try the arm with the highest estimated reward she might regret not exploring other arms whose reward is underestimated but is actually higher than the reward of the arm with highest estimated reward.

A basic question in this context is whether $R(t)$ can be made to grow sub-linearly. Robbins (1952) answered this question in the affirmative. It was later proved (Lai and Robbins 1985) that it is possible in fact to obtain logarithmic regret (the growth of the regret is logarithmic in the number of timesteps). Matching lower bounds (and constants) were also derived.

The Non-stochastic *k*-Armed Bandit Problem

A third popular variant of the bandit problem is the non-stochastic one. In this problem, it is assumed that the sequence of rewards each arm produces is deterministic (possibly adversarial). The decision maker, as in the stochastic bandit problem, wants to minimize her regret, where the regret is measured with respect to the best fixed arm (this best arm might change with time, however). Letting the reward of arm m at time t be $r_m(t)$, we redefine the regret as:

$$r(t) = \max_{1 \le m \le k} \sum_{\tau=1}^{t} r_m(\tau) - \mathbf{E}^{\pi}\left[\sum_{\tau=1}^{t} r(\tau)\right], \quad (2)$$

where the expectation is now taken with respect to randomness in the arm selection. The basic question here is if the regret can be made to grow sub-linearly. The case where the reward of each arm is observed was addressed in the 1950s (see Cesa-Bianchi and Lugosi (2006), for a discussion), where it was shown that there are algorithms that guarantee that the regret grows like $\sqrt{t}$. For the more difficult case, where only the reward of the selected arm is observed and that the rewards of the other arms may not be observed it was shown (Auer et al. 2002) that the same conclusion still holds.

It should be noticed that the optimal policy of the decision maker in this adversarial setting is generally randomized. That is, the decision maker has to select an action at random by following some distribution. The reason is that if the action the decision maker takes is deterministic and can be predicted by Nature, then Nature can consistently "give" the decision maker a low reward for the selected arm while "giving" a high reward to all other arms, leading to a linear regret.

There are some interesting relationships between the non-stochastic bandit problem and prediction with expert advice, universal prediction, and learning in games (Cesa-Bianchi and Lugosi 2006).

The Exploratory *k*-Armed Bandit Problem

This bandit variant emphasizes efficient exploration rather than on the exploration–exploitation tradeoff. As in the stochastic bandit problem, the decision maker is given access to k arms where each arm is associated with an independent and identically distributed random variable with unknown statistics. The decision maker's goal is to identify the "best" arm. That is, the decision maker wishes to find the arm with the highest expected reward as quickly as possible.

The exploratory bandit problem is a sequential hypothesis testing problem but with the added complication that the decision maker can choose where to sample next, making it among the simplest active learning problems. In the context of the probably approximate correct (PAC) setup, it was shown (Mannor and Tsitsiklis 2004) that finding the ε-optimal arm (that is, an arm whose expected reward is lower than that of the best arm by at most ε) with probability of at least $1 - \delta$ requires

$$O\left(\frac{k}{\varepsilon^2}\log\left(\frac{1}{\delta}\right)\right)$$

samples on expectation. Moreover, this bound can be obtained (up to multiplicative constants) via an algorithm known as median elimination.

Bandit analyses such as these have played a key role in understanding the efficiency of ▸ reinforcement-learning algorithm as well.

Cross-References

▸ Active Learning
▸ Associative Bandit Problem
▸ Dynamic Programming

▶ Machine Learning and Game Playing
▶ Markov Process
▶ PAC Learning
▶ Reinforcement Learning

Recommended Reading

Auer P, Cesa-Bianchi N, Freund Y, Schapire RE (2002) The non-stochastic multi-armed bandit problem. SIAM J Comput 32(1):48–77

Berry D, Fristedt B (1985) Bandit problems: sequential allocation of experiments. Chapman and Hall, London/New York

Cesa-Bianchi N, Lugosi G (2006) Prediction, learning, and games. Cambridge University Press, New York

Gittins JC (1989) Multi-armed bandit allocation indices. Wiley, New York

Gittins J, Jones D (1972) A dynamic allocation index for sequential design of experiments. In: Progress in statistics, European meeting of statisticians, Budapest, vol 1, pp 241–266

Lai TL, Robbins H (1985) Asymptotically efficient adaptive allocation rules. Adv Appl Math 6:4–22

Mannor S, Tsitsiklis JN (2004) The sample complexity of exploration in the multi-armed bandit problem. J Mach Learn Res 5:623–648

Robbins H (1952) Some aspects of the sequential design of experiments. Bull Am Math Soc 55:527–535

Kernel Density Estimation

▶ Density Estimation

Kernel Matrix

Synonyms

Gram matrix

Definition

Given a kernel function $k\colon X \times X \to$ and patterns $x_1, \ldots, x_m \in X$, the $m \times m$ matrix K with elements $K_{ij} := k(x_i, x_j)$ is called the kernel matrix of k with respect to $x_1, \ldots, x_m$.

Kernel Methods

Xinhua Zhang
NICTA, Australian National University,
Canberra, ACT, Australia
School of Computer Science, Australian
National University, Canberra, ACT, Australia
NICTA London Circuit, Canberra, ACT,
Australia

Abstract

Over the past decade, kernel methods have gained much popularity in machine learning. Linear estimators have been popular due to their convenience in analysis and computation. However, nonlinear dependencies exist intrinsically in many real applications and are indispensable for effective modeling. Kernel methods can sometimes offer the best of both aspects. The reproducing kernel Hilbert space provides a convenient way to model nonlinearity, while the estimation is kept linear. Kernels also offer significant flexibility in analyzing generic non-Euclidean objects such as graphs, sets, and dynamic systems. Moreover, kernels induce a rich function space where functional optimization can be performed efficiently. Furthermore, kernels have also been used to define statistical models via exponential families or Gaussian processes and can be factorized by graphical models. Indeed, kernel methods have been widely used in almost all tasks in machine learning.

Definition

Kernel methods refer to a class of techniques that employ positive definite kernels. At an algorithmic level, its basic idea is quite intuitive: implicitly map objects to high-dimensional feature spaces and then directly specify the inner product there. As a more principled interpretation, it formulates learning and estimation problems in a reproducing kernel Hilbert space, which is advantageous in a number of ways:

- It induces a rich feature space and admits a large class of (nonlinear) functions.
- It can be flexibly applied to a wide range of domains including both Euclidean and non-Euclidean spaces.
- Searching in this infinite-dimensional space of functions can be performed efficiently, and one only needs to consider the finite subspace expanded by the data.
- Working in the linear spaces of function lends significant convenience to the construction and analysis of learning algorithms.

Motivation and Background

The reproducing kernel was first studied by Aronszajn (1950). Poggio and Girosi (1990) and Wahba (1990) used kernels for data analysis, and Boser et al. (1992) incorporated kernel functions into maximum margin models. Schölkopf et al. (1998) first used kernels for principal component analysis.

Theory

Positive semi-definite (psd) kernels are the most commonly used type of kernels, and its motivation is as follows. Given two objects x_1, x_2 from a space $\mathcal{X}$ which is not necessarily Euclidean, we map them to a high-dimensional feature space via $\phi(x_1)$ and $\phi(x_2)$, and then compute the inner products there by $k(x_1, x_2) = \langle \phi(x_1)\phi(x_2)\rangle$. In many algorithms, the set $\{x_i\}$ influences learning only via inner products between x_i and x_j; hence it is sufficient to specify $k(x_1, x_2)$ directly without explicitly defining ϕ. This leads to considerable savings in computation, when ϕ ranges in high-dimensional spaces or even infinite-dimensional spaces. Clearly, the function k must satisfy some conditions. For example, as a necessary condition, for any finite number of examples $x_1, \ldots, x_n$ from $\mathcal{X}$, the matrix

$$\begin{aligned} K &:= (k(x_i, x_j))_{i,j} \\ &= (\phi(x_1), \ldots, \phi(x_n))^\top (\phi(x_1), \ldots, \phi(x_n)) \end{aligned}$$

must be positive semi-definite. Surprisingly, this turns out to be a sufficient condition as well, and hence we define the positive semi-definite kernels.

Definition 1 (positive semi-definite kernels) Let $\mathcal{X}$ be a nonempty set. A function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is called a positive semi-definite kernel if for any $n \in \mathbb{N}$ and $x_1, \ldots, x_n \in \mathcal{X}$, the Gram matrix $K := (k(x_i, x_j))_{i,j}$ is symmetric and positive semi-definite (psd).

Reproducing Kernel Hilbert Space

Given a psd kernel k, we are able to construct a map ϕ from $\mathcal{X}$ to an inner product space $\mathcal{H}$, such that $\langle \phi(x_1)\phi(x_2)\rangle = k(x_1, x_2)$. The image of x under ϕ is just a function $\phi(x) := k(x, \cdot)$, where $k(x, \cdot)$ is a function of $\cdot$, assigning the value $k(x, x')$ for any $x' \in \mathcal{X}$. To define inner products between functions, we need to construct an inner product space $\mathcal{H}$ which contains $\{k(x, \cdot) : x \in \mathcal{X}\}$. First, $\mathcal{H}$ must contain the linear combinations $\left\{\sum_{i=1}^{n} \alpha_i k(x_i, \cdot) : n \in \mathbb{N}, x_i \in \mathcal{X}, \alpha_i \in \mathbb{R}\right\}$. Then we endow it with an inner product as follows. For any $f, g \in \mathcal{H}$ and $f = \sum_{i=1}^{n} \alpha_i k(x_i, \cdot)$, $g = \sum_{j=1}^{m} \beta_j k(x'_j, \cdot)$, define

$$\langle fg\rangle := \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j k(x_i, x'_j),$$

and it is easy to show that this is well defined (independent of the expansion of f and g). Using the induced norm, we can complete the space and thus get a Hilbert space $\mathcal{H}$, which is called reproducing kernel Hilbert space (RKHS). The term "reproducing" is because for any function $f \in \mathcal{H}$, $\langle f k(x, \cdot)\rangle = f(x)$.

Properties of psd Kernels

Let $\mathcal{X}$ be a nonempty set and $k_1, k_2, \ldots$ be arbitrary psd kernels on $\mathcal{X} \times \mathcal{X}$. Then

- The set of psd kernels is a closed convex cone, *i.e.*, (a) if $\alpha_1, \alpha_2 \geq 0$, then $\alpha_1 k_1 + \alpha_2 k_2$ is psd and (b) if $k(x, x') := \lim_{n \to \infty} k_n(x, x')$ exists for all x, x', then k is psd.
- The pointwise product $k_1 k_2$ is psd.

- Assume for $i = 1, 2$, k_i is a psd kernel on $\mathcal{X}_i \times \mathcal{X}_i$, where $\mathcal{X}_i$ is a nonempty set. Then the tensor product $k_1 \otimes k_2$ and the direct sum $k_1 \oplus k_2$ are psd kernels on $(\mathcal{X}_1 \times \mathcal{X}_2) \times (\mathcal{X}_1 \times \mathcal{X}_2)$.

Example Kernels

One of the key advantages of kernels lies in its applicability to a wide range of objects.

Euclidean Spaces In $\mathbb{R}^n$, popular kernels include linear kernel $k(x_1, x_2) = \langle x_1 x_2 \rangle$, polynomial kernels $k(x_1, x_2) = (\langle x_1 x_2 \rangle + c)^d$ where $d \in \mathbb{N}$ and $c \geq 0$, Gaussian RBF kernels $k(x_1, x_2) = \exp(-\gamma \|x_1 - x_2\|^2)$ where $\gamma > 0$, and Laplacian RBF kernels $k(x_1, x_2) = \exp(-\gamma \|x_1 - x_2\|)$. Another useful type of kernels on Euclidean spaces is the spline kernels.

Convolution Kernels Haussler (1999) investigated how to define a kernel between composite objects by building on the similarity measures that assess their respective parts. It needs to enumerate all possible ways to decompose the objects; hence efficient algorithms like dynamic programming are needed.

Graph Kernels Graph kernels are available in two categories: between graphs and on a graph. The first type is similar to convolution kernels, which measures the similarity between two graphs. The second type defines a metric between the vertices and is generally based on the graph Laplacian. By applying various transform functions to the eigenvalue of the graph Laplacian, various smoothing and regularization effects can be achieved.

Fisher Kernels Kernels can also be defined between probability densities $p(x|\theta)$. Let $U_\theta(x) = -\partial_\theta \log p(x|\theta)$ and $I = \mathbb{E}_x[U_\theta(x) U_\theta^\top(x)]$ be the Fisher score and Fisher information matrix, respectively. Then the normalized and unnormalized Fisher kernels are defined by

$$k(x, x') = U_\theta^\top(x) I^{-1} U_\theta(x')$$
$$\text{and} \quad k(x, x') = U_\theta^\top(x) U_\theta(x'),$$

respectively. In theory, estimation using normalized Fisher kernels corresponds to regularization on the $L_2(p(\cdot|\theta))$ norm. And in the context of exponential families, the unnormalized Fisher kernels are identical to the inner product of sufficient statistics.

Kernel Function Classes

Many machine learning algorithms can be posed as functional minimization problems, and the candidate function set is chosen as the RKHS. The main advantage of optimizing over an RKHS originates from the representer theorem.

Theorem 2 (representer theorem) *Denote by $\Omega : [0, \infty) \mapsto \mathbb{R}$ a strictly monotonic increasing function, by $\mathcal{X}$ a set, and by $c : (\mathcal{X} \times \mathbb{R}^2)^n \mapsto \mathbb{R} \cup \{\infty\}$ an arbitrary loss function. Then each minimizer $f \in \mathcal{H}$ of the regularized risk functional*

$$c((x_1, y_1, f(x_1)), \ldots, (x_n, y_n, f(x_n))) + \Omega(\|f\|_{\mathcal{H}}^2) \quad (1)$$

admits a representation of the form

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x).$$

The representer theorem is important in that although the optimization problem is in an infinite-dimensional space $\mathcal{H}$, the solution is guaranteed to lie in the span of n particular kernels centered on the training points.

The objective (1) is composed of two parts: the first part measures the loss on the training set $\{x_i, y_i\}_{i=1}^n$ which depends on f only via its value at x_i. The second part is the regularizer, which encourages small RKHS norm of f. Intuitively, this regularizer penalizes the complexity of f and prefers smooth f. When the kernel k is translation invariant, *i.e.*, $k(x_1, x_2) = h(x_1 - x_2)$, Smola et al. (1998) showed that $\|f\|^2$ is related to the Fourier transform of h, with more penalty imposed on the high-frequency components of f.

Applications

Kernels have been applied to almost all branches of machine learning.

Supervised Learning

One of the most well-known applications of kernel method is the SVM for binary classification. Its primal form can be written as

$$\underset{w,b,\xi}{\text{minimize}} \frac{\lambda}{2}\,\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n}\xi_i,$$

$$s.t.\ y_i(\langle w x_i\rangle + b) \geq 1 - \xi_i,\ \text{and}\ \xi_i \geq 0,\ \forall i.$$

Its dual form can be written as

$$\underset{\alpha_i}{\text{minimize}} \frac{1}{2\lambda}\sum_{i,j} y_i y_j \left\langle x_i x_j\right\rangle \alpha_i \alpha_j - \sum_i \alpha_i,$$

$$s.t.\ \sum_i y_i \alpha_i = 0,\ \alpha_i \in [0, n^{-1}],\ \forall i.$$

Clearly, this can be extended to feature maps and kernels by setting $k(x_i, x_j) = \left\langle x_i x_j\right\rangle$. The same trick can be applied to other algorithms like ν-SVM, regression, density estimation, etc. For multi-class classification and structured output classification where the possible label set $\mathcal{Y}$ can be large, kernel maximum margin machines can be formulated by introducing a joint kernel on pairs of (x_i, y) $(y \in \mathcal{Y})$, i.e., the feature map takes the tuple (x_i, y). Letting $\Delta(y_i, y)$ be the discrepancy between the true label y_i and the candidate label y, the primal form is

$$\underset{w,\xi_i}{\text{minimize}} \frac{\lambda}{2}\,\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n}\xi_i, \quad s.t.\ \big\langle w\phi(x_i, y_i) - \phi(x_i, y)\big\rangle \geq \Delta(y_i, y) - \xi_i,\ \forall\ i, y,$$

and the dual form is

$$\underset{\alpha_{i,y}}{\text{minimize}} \frac{1}{2\lambda}\sum_{(i,y),(i',y')} \alpha_{i,y}\alpha_{i',y'}\big\langle \phi(x_i, y_i) - \phi(x_i, y)\phi(x_{i'}, y_{i'}) - \phi(x_{i'}, y')\big\rangle - \sum_{i,y}\Delta(y_i, y)\alpha_{i,y}$$

$$s.t.\quad \alpha_{i,y} \geq 0,\ \forall\ i, y;\quad \sum_y \alpha_{i,y} = \frac{1}{n},\ \forall i.$$

Again all the inner products $\langle \phi(x_i, y)\phi(x_{i'}, y')\rangle$ can be replaced by the joint kernel $k((x_i, y), (x_{i'}, y'))$. Further factorization using graphical models is possible; see Taskar et al. (2004). Notice when $\mathcal{Y} = \{1, -1\}$, setting $\phi(x_i, y) = y\phi(x_i)$ recovers the binary SVM formulation. Effective methods to optimize the dual objective include sequential minimal optimization, exponentiated gradient (Collins et al. 2008), mirror descent, cutting plane, or bundle methods (Smola et al. 2007a).

Unsupervised Learning

Data analysis can benefit from modeling the distribution of data in feature space. There we can still use the rather simple linear methods, which give rise to nonlinear methods on the original data space. For example, the principal components analysis (PCA) can be extended to Hilbert spaces (Schölkopf et al. 1998), which allows for image denoising, clustering, and nonlinear dimensionality reduction.

Given a set of data points $\{x_i\}_{i=1}^{n}$, PCA tries to find a direction d such that the projection of $\{x_i\}$ to d has the maximal variance. Mathematically, one solves

$$\max_{d:\|d\|=1} \operatorname{Var}\{\langle x_i d\rangle\} \iff \max_{d:\|d\|=1} d^\top \times \left(\frac{1}{n}\sum_i x_i x_i^\top - \frac{1}{n^2}\sum_{ij} x_i x_j^\top\right) d,$$

which can be solved by finding the maximum eigenvalue of the variance of $\{x_i\}$. Along the same line, we can map the examples to the RKHS and find the maximum variance projection direction again. Here we first center the data, i.e., let the feature map be $\tilde{\phi}(x_i) = \phi(x_i) - \frac{1}{n}\sum_j \phi(x_j)$, and define a kernel $\tilde{k}$ based on the centered feature. So we have $\sum_{j=1}^{n}\tilde{K}_{ij} = 0$ for all i. Now the objective can be written as

$$\max_{f:\|f\|_{\tilde{\mathcal{H}}}=1} \operatorname{Var}\left\{\big\langle \tilde{\phi}(x_i) f\big\rangle_{\tilde{\mathcal{H}}}\right\} \iff \max_{f:\|f\|=1} \operatorname{Var}\{f(x_i)\} \iff \max_{f:\|f\|\leq 1} \operatorname{Var}\{f(x_i)\}. \quad (2)$$

Treat the constraint $\|f\| \leq 1$ as an indicator function $\Omega(\|f\|^2)$ where $\Omega(x) = 0$ if $x \leq 1$ and ∞ otherwise. Then the representer theorem can be invoked to guarantee that the optimal solution is $f = \sum_i \alpha_i \tilde{k}(x_i, \cdot)$ for some $\alpha_i \in \mathbb{R}$. Plugging it into (2), the problem becomes $\max_{\boldsymbol{\alpha}:\boldsymbol{\alpha}^\top \tilde{K}\boldsymbol{\alpha}=1} \boldsymbol{\alpha}^\top \tilde{K}^2\boldsymbol{\alpha}$. To get necessary conditions for optimality, we write out the Lagrangian $L = \boldsymbol{\alpha}\tilde{K}^2\boldsymbol{\alpha} - \lambda(\boldsymbol{\alpha}\tilde{K}\boldsymbol{\alpha} - 1)$. Setting to 0 the derivative over $\boldsymbol{\alpha}$, we get

$$\tilde{K}^2\boldsymbol{\alpha} = \lambda\tilde{K}\boldsymbol{\alpha}. \tag{3}$$

Therefore $\boldsymbol{\alpha}^\top \tilde{K}^2\boldsymbol{\alpha} = \lambda$. Although (3) does not guarantee that $\boldsymbol{\alpha}$ is an eigenvector of $\tilde{K}$, one can show that for each λ satisfying (3), there exists an eigenvector $\boldsymbol{\alpha}$ of $\tilde{K}$ such that $\tilde{K}\boldsymbol{\alpha} = \lambda\boldsymbol{\alpha}$. Hence, it is sufficient to study the eigensystem of $\tilde{K}$ just like in the vanilla PCA. Once the optimal α_i^* is obtained, any data point x can be projected to $\sum_i \alpha_i^* \tilde{k}(x_i, x)$.

More applications of kernels in unsupervised learning can be found in canonical correlation analysis, independent component analysis (Bach and Jordan 2002), kernelized independence criteria via Hilbert space embeddings of distributions (Smola et al. 2007b), etc.

Cross-References

▶ Principal Component Analysis
▶ Radial Basis Function Networks
▶ Support Vector Machines

Recommended Reading

A survey paper on kernel methods up to year 2007 is Hofmann et al. (2008). For an introduction to SVMs and kernel methods, read Cristianini and Shawe-Taylor (2000). More comprehensive treatment can be found in Schölkopf and Smola (2002), Shawe-Taylor and Cristianini (2004), and Steinwart and Christmann (2008). As far as applications are concerned, see Lampert (2009) for computer vision and Schölkopf et al. (2004) for bioinformatics. Finally, Vapnik (1998) provides the details on statistical learning theory.

Aronszajn N (1950) Theory of reproducing kernels. Trans Am Math Soc 68:337–404

Bach FR, Jordan MI (2002) Kernel independent component analysis. J Mach Learn Res 3:1–48

Boser B, Guyon I, Vapnik V (1992) A training algorithm for optimal margin classifiers. In: Haussler D (ed) Proceedings of annual conference on computational learning theory. ACM Press, Pittsburgh, pp 144–152

Collins M, Globerson A, Koo T, Carreras X, Bartlett P (2008) Exponentiated gradient algorithms for conditional random fields and max-margin markov networks. J Mach Learn Res 9:1775–1822

Cristianini N, Shawe-Taylor J (2000) An introduction to support vector machines and other kernel-based learning methods. Cambridge University Press, Cambridge

Haussler D (1999) Convolution kernels on discrete structures. Technical report UCS-CRL-99-10, UC Santa Cruz

Hofmann T, Schölkopf B, Smola AJ (2008) Kernel methods in machine learning. Ann Stat 36(3):1171–1220

Lampert CH (2009) Kernel methods in computer vision. Found Trends Comput Graph Vis 4(3): 193–285

Poggio T, Girosi F (1990) Networks for approximation and learning. Proc IEEE 78(9):1481–1497

Schölkopf B, Smola A (2002) Learning with kernels. MIT Press, Cambridge

Schölkopf B, Smola AJ, Müller K-R (1998) Nonlinear component analysis as a kernel eigenvalue problem. Neural Comput 10:1299–1319

Schölkopf B, Tsuda K, Vert J-P (2004) Kernel methods in computational biology. MIT Press, Cambridge

Shawe-Taylor J, Cristianini N (2004) Kernel methods for pattern analysis. Cambridge University Press, Cambridge

Smola A, Vishwanathan SVN, Le Q (2007a) Bundle methods for machine learning. In: Koller D, Singer Y (eds) Advances in neural information processing systems, vol 20. MIT Press, Cambridge

Smola AJ, Gretton A, Song L, Schölkopf B (2007b) A Hilbert space embedding for distributions. In: International conference on algorithmic learning theory, Sendai. Volume 4754 of LNAI. Springer, pp 13–31

Smola AJ, Schölkopf B, Müller K-R (1998) The connection between regularization operators and support vector kernels. Neural Netw 11(5): 637–649

Steinwart I, Christmann A (2008) Support vector machines. Information science and statistics. Springer, New York

Taskar B, Guestrin C, Koller D (2004) Max-margin Markov networks. In: Thrun S, Saul L, Schölkopf B (eds) Advances in neural information processing systems, vol 16. MIT Press, Cambridge, pp 25–32

Vapnik V (1998) Statistical learning theory. Wiley, New York

Wahba G (1990) Spline models for observational data. Volume 59 of CBMS-NSF regional conference series in applied mathematics. SIAM, Philadelphia

Kernel Shaping

▶ Local Distance Metric Adaptation ▶ Locally Weighted Regression for Control

Kernel-Based Reinforcement Learning

▶ Instance-Based Reinforcement Learning

Kernels

▶ Gaussian Process

Kind

▶ Class

K-Means Clustering

Xin Jin[1] and Jiawei Han[2]
[1]PayPal Inc., San Jose, CA, USA
[2]University of Illinois at Urbana-Champaign, Urbana, IL, USA

Abstract

K-Means Clustering is a popular clustering algorithm with local optimization. In order to improve its performance, researchers have proposed methods for better initialization and faster computation.

Synonyms

Cluster initialization; Iterative computation

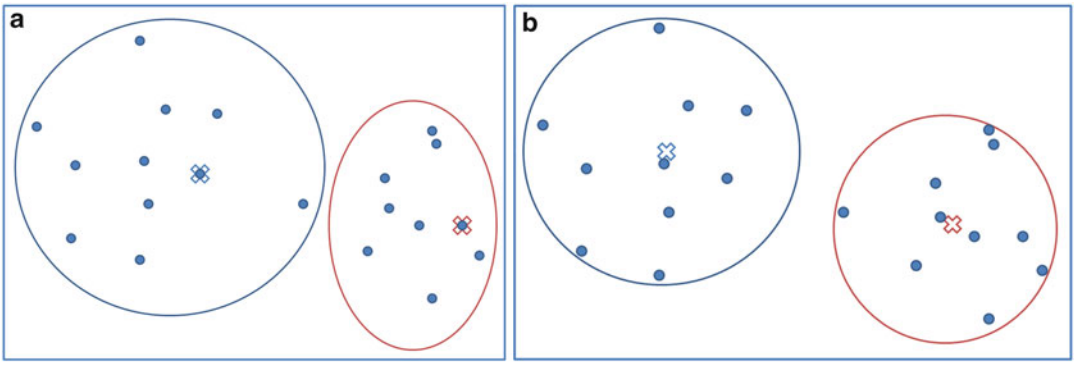
Definition

K-means (Lloyd 1957; MacQueen 1967) is a popular data clustering method, widely used in many applications. Algorithm 1 shows the procedure of K-means clustering. The basic idea of the K-means clustering is that given an initial but not optimal clustering, relocate each point to its new nearest center, update the clustering centers by calculating the mean of the member points, and repeat the relocating-and-updating process until converge criteria (such as predefined number of iterations, difference on the value of the distortion function) are satisfied.

The task of initialization is to form the initial K clusters. Many initializing techniques have been proposed, from simple methods, such as choosing the first K data points, Forgy initialization (randomly choosing K data points in the dataset), and random partitions (dividing the data points randomly into K subsets), to more sophisticated methods, such as density-based initialization, intelligent initialization, furthest-first initialization (FF for short, it works by picking first center point randomly and then adding more center points which are furthest from existing ones), and subset furthest-first (SFF) initialization. For more details, refer to paper Steinley and Brusco (2007) which provides a survey and comparison of over 12 initialization methods.

Algorithm 1 K-means clustering algorithm

```
Input: $K$, number of clusters; $D$, a data set of $N$ points
Output: A set of $K$ clusters
Initialization.
repeat
    for each point $p$ in $D$ do
        find the nearest center and assign $p$ to the corresponding cluster.
    end for
    update clusters by calculating new centers using mean of the members.
until stop-iteration criteria satisfied
return clustering result.
```



***K*-Means Clustering, Fig. 1** K-means clustering example ($K = 2$). The center of each cluster is marked by "x." (**a**) Initialization. (**b**) Re-assignment

Figure 1 shows an example of K-means clustering on a set of points, with $K = 2$. The clusters are initialized by randomly selecting two points as centers.

Complexity analysis. Let N be the number of points, D the number of dimensions, and K the number of centers. Suppose the algorithm runs I iterations to converge. The space complexity of K-means clustering algorithm is $O(N(D + K))$. Based on the number of distance calculations, the time complexity of K-means is $O(NKI)$.

Fast Computation for Large-Scale Data

For large-scale data clustering, K-means algorithm spends the majority of the time on the numerous distance calculations between the points and the centers. Many algorithms have been proposed to handle this problem, such as PDS (Bei and Gray 1985), TIE (Chen and Hsieh 1991), Elkan (Beil et al. 2003), MPS (Ra and Kim 1993), kd-tree K-means (Pelleg and Moore 1999), HKM (Nister and Stewenius 2006), GT (Kaukoranta et al. 2000), CGAUDC (Lai et al. 2008), and GAD (Jin et al. 2011).

PDS (partial distortion search) (Bei and Gray 1985) cumulatively computes the distance between the point and a candidate center by summing up the differences at each dimension. TIE (triangular inequality elimination) (Chen and Hsieh 1991) prunes candidate centers based on the triangle inequality of metric distance. MPS (mean-distance-ordered partial search) (Ra and Kim 1993) uses sorting to initially guess the center whose mean value is closest to that of the current point and prune candidates via an inequality based on an Euclidean distance property.

In many large-scale applications, we need to perform large K clustering, and HKM (Nister and Stewenius 2006) and kd-tree K-means (Pelleg and Moore 1999) are fast algorithms which work for this *large cluster* problem because their time complexity on K is reduced from the original $O(K)$ in K-means to $O(\log(K))$.

HKM (Nister and Stewenius 2006) performs fast hierarchical K-means clustering. Instead of directly performing clustering on large clusters, HKM uses K-means for a small number of clusters at each node of a hierarchical tree.

The kd-tree K-means (Pelleg and Moore 1999) algorithm utilizes kd-tree to find the approximate nearest center in a way that is faster than brute force searching. Centers were split hierarchically from the root to the leaf nodes of the kd-tree; leaf nodes will contain similar centers. When searching for the nearest center, we only need to check leaf nodes which are most similar to the point.

Many fast algorithms are based on a strategy that filters out unnecessary distance calculations using metric properties and thus only work for metric distances. Another strategy called *activity* *detection* avoids the metric properties and works for both metric and non-metric distances. GT (Kaukoranta et al. 2000) utilizes point activity for fast clustering. CGAUDC (Lai et al. 2008) is an extension of GT and gets further improvement on performance. GAD (Jin et al. 2011) provides a general solution for utilizing activity detection for fast clustering.

Software

The following software have implementations of the K-means clustering algorithm:

- Weka. Open source data mining software in Java (Hall et al. 2009), from Machine Learning Group at the University of Waikato:
  http://www.cs.waikato.ac.nz/ml/weka/index.html
- Apache Mahout. Open source machine learning software in Java for use in Hadoop, with support on K-means, Fuzzy K-means, and streaming K-means:
  http://mahout.apache.org/users/clustering/k-means-clustering.html
- LNKnet Software. Written in C. A public domain software from MIT Lincoln Laboratory:
  http://www.ll.mit.edu/mission/communications/cyber/softwaretools/lnknet/lnknet.html
- R K-means. R package. It performs K-means clustering on a data matrix.
  http://stat.ethz.ch/R-manual/R-patched/library/stats/html/kmeans.html
- MLPack. A scalable C++ machine learning library.
  http://mlpack.org
- Scikit-Learn. An open source machine learning software written in Python.
  http://scikit-learn.org

Recommended Reading

Bei C-D, Gray RM (1985) An improvement of the minimum distortion encoding algorithm for vector quantization. IEEE Trans Commun 33:1132–1133

Beil F, Ester M, Xu X (2003) Using the triangle inequality to accelerate k-means. In: Twentieth international conference on machine learning (ICML'03), Washington, DC, pp 147–153

Chen S-H, Hsieh WM (1991) Fast algorithm for VQ codebook design. In: IEE Proceedings I-Communications, Speech and Vision, 138(5):357–362

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The weka data mining software: an update. ACM SIGKDD Explor Newsl 11(1): 10–18

Jin X, Kim S, Han J, Cao L, Yin Z (2011) A general framework for efficient clustering of large datasets based on activity detection. Stat Anal Data Min 4(1):11–29

Kaukoranta T, Franti P, Nevalainen O (2000) A fast exact gla based code vector activity detection. IEEE Trans Image Process 9(8):1337–1342

Lai JZC et al (2008) A fast VQ codebook generation algorithm using codeword displacement. Pattern Recognit 41(1):315–319

Lloyd SP (1957) Least squares quantization in pcm. Technical report RR-5497, Bell Lab, Sept 1957

MacQueen JB (1967) Some methods for classification and analysis of multivariate observations. In: Le Cam LM, Neyman J (eds) Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, vol 1. University of California Press, Berkeley, pp 281–297

Nister D, Stewenius H (2006) Scalable recognition with a vocabulary tree. In: CVPR06, New York

Pelleg D, Moore A (1999) Accelerating exact k-means algorithms with geometric reasoning. In: Proceedings of KDD'99, New York. ACM, pp 277–281

Ra S-W, Kim J-K (1993) A fast mean-distance-ordered partial codebook search algorithm for image vector quantization. IEEE Trans Circuits Syst 40:576–579

Steinley D, Brusco MJ (2007) Initializing k-means batch clustering: a critical evaluation of several techniques. J Classif 24(1):99–121

K-Medoids Clustering

Xin Jin[1] and Jiawei Han[2]
[1]PayPal Inc., San Jose, CA, USA
[2]University of Illinios at Urbana-Champaign, Urbana, IL, USA

Abstract

K-Medoids Clustering is a clustering method more robust to outliers than K-Means. Representative algorithms include Partitioning Around Medoids (PAM), CLARA, CLARANS, etc.

Synonyms

Iterative algorithm

Definition

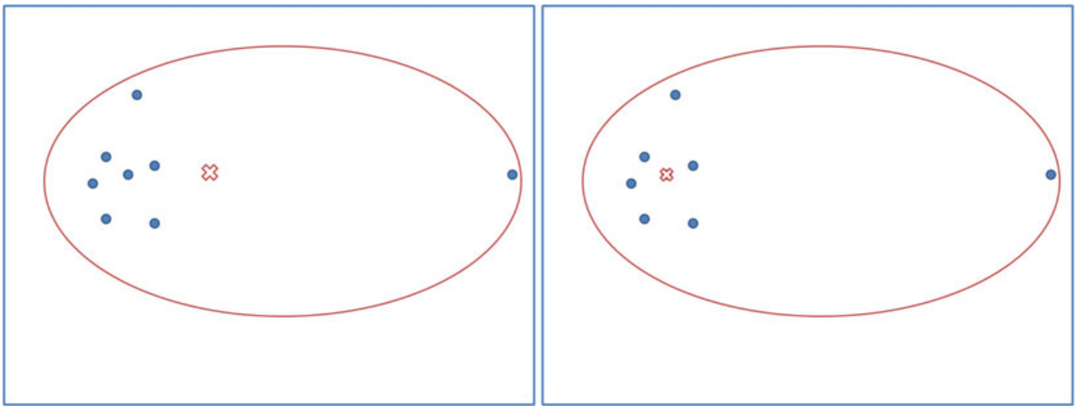
The K-means clustering algorithm is sensitive to outliers, because a mean is easily influenced by extreme values. K-medoids clustering is a variance of K-means but more robust to noises and outliers (Han et al. 2011). Instead of using the mean point as the center of a cluster, K-medoids use an actual point in the cluster to represent it. Medoid is the most centrally located object of the cluster, with minimum sum of distances to other points. Figure 1 shows the difference between mean and medoid in a 2D example. The group of points in the right form a cluster, while the rightmost point is an outlier. Mean is greatly influenced by the outlier and thus cannot represent the correct cluster center, while medoid is robust to the outlier and correctly represents the cluster center.

Partitioning Around Medoids (PAM) (Kaufman and Rousseeuw 2005) is a representative K-medoids clustering method. The basic idea is as follows: select K representative points to form initial clusters and then repeatedly move to better cluster representatives. All possible combinations of representative and nonrepresentative points are analyzed, and the quality of the resulting clustering is calculated for each pair. An original representative point is replaced with the new point which causes the greatest reduction in distortion function. At each iteration, the set of best points for each cluster form the new respective medoids.

When calculating the cost of swapping a nonrepresentative point p_{rand} with a representative point p_i, there are four cases that need to be examined for each of the nonrepresentative point p.

1. Case 1: p originally belongs to representative point p_i. If, after replacement, p is closest to one of the other representative point p_j, then p is reassigned to p_j.
2. Case 2: p originally belongs to representative point p_i. If after replacement, p is closest to p_{rand}, then p is reassigned to p_{rand}.
3. Case 3: p originally belongs to one of the other representative point p_j. If after replacement, p is still closest to p_j, then there is no reassignment of p.
4. Case 4: p originally belongs to one of the other representative point p_j. If after replacement, p is closest to p_{rand}, then p is reassigned to p_{rand}.

The cost function is defined as the change in the value of the distortion function when a representative point is replaced by a nonrepresentative point. The total cost C of replacing is the sum of costs incurred by all nonrepresentative points.



K-Medoids Clustering, Fig. 1 Mean vs. medoid in 2D space. In both figures (**a**) and (**b**), the group of points in the right forms a cluster and the rightmost point is an outlier. *The red point* represents the center found by mean or medoid

If C is negative, then the replacement is allowed since the distortion function would be reduced.

Algorithm 1 K-medoids clustering algorithm (PAM)

```
Require: $K$, number of clusters; $D$, a data set of $N$ points
Ensure: A set of $K$ clusters
Arbitarily choose $K$ points in $D$ as initial representative points.
repeat
    for each non-representative point $p$ in $D$ do
        find the nearest representative point and assign $p$ to the corresponding cluster.
    end for
    randomly select a non-representative point $p_{\text{rand}}$;
    compute the overall cost $C$ of swapping a representative point $p_i$ with $p_{\text{rand}}$;
    if $C < 0$ then
        swap $p_j$ with $p_{\text{rand}}$ to form a new set of $K$ representative points.
    end if
until stop-iteration criteria satisfied
return clustering result.
```

The time complexity of the PAM algorithm is $O(K(N - K)^2 I)$.

Fast Computation for Large Data

PAM is not scalable for large data set; some algorithms have been proposed to improve the efficiency, such as CLARA (clustering large applications) (Kaufman and Rousseeuw 2005) and CLARANS (clustering large applications based upon randomized search) (Ng and Han 2002).

CLARA takes the sampling strategy: randomly sample a small portion of the actual data as a representative of the whole data and perform PAM from the sampled data set to find the K medoids. If the sample can closely represent the original data set, the representative medoids found will be a good approximation of those that found from using the whole data set. To improve clustering quality, CLARA takes multiple random samples, applies PAM on each sample, and outputs the best clustering. Suppose the size of the sample is M, which is much smaller than the original data size N; the time complexity of CLARA is $O((KM^2 + K(N - K))I)$. CLARA is more efficient than PAM; however, it cannot find the best clustering if any medoid found from the sample is not among the best K-medoids.

CLARANS is able to improve the quality of CLARA. It can be modeled as searching through a graph where a node is a set of K-medoids and neighbor nodes differ by one medoid, so each node has $K(N - K)$ neighbors. A node is evaluated by the distortion function to measure its clustering quality. CLARANS starts with a randomly selected node and randomly selects its neighbor; if the neighbor has better clustering quality, move to the neighbor node and continue iteration. If the node is a local minimum, i.e., no tested neighbor gets better clustering, restart with a new randomly selected node and repeat the procedure. Iteration stops at some criteria, for example, finding a predefined number of local minima. Some improvements (Chu et al. 2008) have been proposed for CLARANS. There are two categories, conceptual/algorithmic and implementational improvements, including the revisiting of the accepted cases for swap comparison and the application of partial distance searching and previous medoid indexing to clustering.

Softwares

The following softwares have implementations of the K-medoids clustering algorithm:

- Flexclust: flexible cluster algorithms. R package. It provides a general framework for k-centroids cluster analysis supporting arbitrary distance measures and centroid computation. http://cran.r-project.org/web/packages/flexclust/index.html
- Julia. Clustering package: https://github.com/JuliaStats/Clustering.jl
- ELKI (for environment for developing KDD-applications supported by index structures). A Java-based data mining software framework developed at the Ludwig Maximilian University of Munich, Germany. http://elki.dbs.ifi.lmu.de

- Java-ML. Java Machine Learning Library: http://java-ml.sourceforge.net

Recommended Reading

Chu S-C, Roddick JF, Pan J-S (2008) Improved search strategies and extensions to k-medoids-based clustering algorithms. Int J Bus Intell Data Min 3(2):212–231

Han J, Kamber M, Pei J (2011) Data mining: concepts and techniques, 3rd edn. Morgan Kaufmann Publishers, San Francisco

Kaufman L, Rousseeuw PJ (2005) Finding groups in data: an introduction to cluster analysis. Wiley series in probability and statistics. Wiley-Interscience, New York

Ng RT, Han J (2002) CLARANS: a method for clustering objects for spatial data mining. IEEE Trans Knowl Data Eng 14(5):1003–1016

Kohonen Maps

▸ Self-Organizing Maps

K-Way Spectral Clustering

Xin Jin[1] and Jiawei Han[2]
[1]PayPal Inc., San Jose, CA, USA
[2]University of Illinois at Urbana-Champaign,
Urbana, IL, USA

Abstract

K-Way Spectral Clustering is the technology to discover k clusters using spectral clustering.

Synonyms

Diagonal matrix; Eigenvector; Laplacian matrix; Spectral clustering

Definition

In spectral clustering (Luxburg 2007; Zha et al. 2001; Dhillon et al. 2004), the dataset is represented as a similarity graph $G = (V, E)$. The vertices represent the data points. Two vertices are connected if the similarity between the corresponding data points is larger than a certain threshold, and the edge is weighted by the similarity value. Clustering is achieved by choosing a suitable partition of the graph that each group corresponds to one cluster.

A good partition (i.e., a good clustering) is that the edges between different groups have an overall low weight and the edges within a group have high weight, which indicates that points in different clusters are dissimilar from each other and points within the same cluster are similar to each other. One basic spectral clustering algorithm finds a good partition in the following way:

Given a set of data points P and the similarity matrix S where S_{ij} measures the similarity between points $i, j \in P$, form a graph. Build Laplacian matrix L of the graph,

$$L = I - D^{-1/2} S D^{-1/2} \qquad (1)$$

where D is the diagonal matrix:

$$D_{ii} = \sum_j S_{ij} \qquad (2)$$

Find eigenvalues and eigenvectors of the matrix L, map vertices to corresponding components, and form clusters based on the embedding space.

The methods to find K clusters include recursive bi-partitioning and clustering multiple eigenvectors. The former technique is inefficient and unstable. The latter approach is more preferable because it is able to prevent instability due to information loss.

Recommended Reading

Dhillon IS, Guan Y, Kulis B (2004) Kernel k-means: spectral clustering and normalized cuts. In: Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, pp 551–556

Luxburg U (2007) A tutorial on spectral clustering. Stat Comput 17(4):395–416

Zha H, He X, Ding C, Gu M, Simon HD (2001) Spectral relaxation for k-means clustering. In: Advances in neural information processing systems, Neural Information Processing Systems (NIPS), pp 1057–1064

L

L_1-Distance

▶ Manhattan Distance

Label

A label is a target value that is associated with each ▶ object in ▶ training data. In ▶ classification learning, labels are ▶ classes. In ▶ regression, labels are numeric.

Labeled Data

Labeled data are data for which each ▶ object has an identified target value, the ▶ label. Labeled data are used in ▶ supervised learning. They stand in contrast to *unlabeled data* that are used in ▶ unsupervised learning.

Language Bias

Definition

A learner's language bias is the set of hypotheses that can be expressed using the hypothesis language employed by the learner.

This language bias can be implicit, or it can be defined explicitly, using a bias specification language (see ▶ Bias Specification Language).

Cross-References

▶ Learning as Search

Laplace Estimate

▶ Gaussian Process

Laplacian Matrix

▶ *K*-Way Spectral Clustering

Latent Class Model

▶ Mixture Model

Latent Factor Models and Matrix Factorizations

Definition

Latent Factor models are a state of the art methodology for model-based ▶ collaborative filtering. The basic assumption is that there exist an unknown low-dimensional representation of users and items where user-item affinity can be modeled accurately. For example, the rating that a user gives to a movie might be assumed to depend on few implicit factors such as the user's taste across various movie genres. Matrix

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

factorization techniques are a class of widely successful Latent Factor models that attempt to find weighted low-rank approximations to the user-item matrix, where weights are used to hold out missing entries. There is a large family of matrix factorization models based on choice of loss function to measure approximation quality, regularization terms to avoid overfitting, and other domain-dependent formulations.

Lazy Learning

Geoffrey I. Webb
Faculty of Information Technology, Monash
University, Victoria, Australia

Abstract

Lazy learning is a machine learning strategy whereby learning is delayed until consultation time.

Definition

The computation undertaken by a learning system can be viewed as occurring at two distinct times, *training time* and *consultation* or *testing time*. Consultation time is the time between when an object is presented to a system for an inference to be made and the time when the inference is completed. Training time is time prior to consultation time during which the system makes inferences from training data in preparation for consultation time. *Lazy learning* refers to any machine learning process that defers the majority of computation to consultation time. Two typical examples of lazy learning are ▶ instance-based learning and Lazy Bayesian Rules.

Lazy learning stands in contrast to *eager learning*, in which the majority of computation occurs at training time.

Discussion

Lazy learning can be computationally advantageous when predictions using a single training set will only be made for few objects. This is because it is only necessary to model the immediate areas of the ▶ instance space that are occupied by objects to be classified. In consequence, no computation is expended modeling areas of the instance space that are irrelevant to the predictions that need to be made. This can also be an advantage when a training set is frequently updated, as can be the case in ▶ online learning, as it is not necessary to create a complete global model before making a prediction subsequent to new training examples becoming available.

Lazy learning can help improve prediction ▶ accuracy, by allowing a system to concentrate on deriving the best possible decision for the exact points of the instance space for which predictions are to be made. In contrast, eager learning can sometimes result in suboptimal predictions for some specific areas of the instance space as a result of trade-offs during the process of deriving a single model that seeks to minimize average error over the entire instance space.

However, lazy learning must store the entire training set for use in classification. In contrast, eager learning need only store a model, which may be must more compact than the original data.

Cross-References

▶ Instance-Based Learning
▶ Locally Weighted Regression for Control
▶ Online Learning

References

Aha, David W. Lazy learning. Kluwer academic publishers, 1997.

Learning Algorithm Evaluation

▶ Evaluation of Learning Algorithms

Learning as Search

Claude Sammut
The University of New South Wales, Sydney,
NSW, Australia

Definition

Learning can be viewed as a search through the space of all sentences in a concept description language for a sentence that best describes the data. Alternatively, it can be viewed as a search through all hypotheses in a ▶ hypothesis space. In either case, a generality relation usually determines the structure of the search space.

Background

The input to a learning program consists of descriptions of objects from the universe (the ▶ training set) and, in the case of ▶ supervised learning, an output value associated with the example. A program is limited in the concepts that it can learn by the representational capabilities of both the ▶ observation language (i.e., the language used to describe the training examples) and ▶ hypothesis language (the language used to describe the concept). For example, if an attribute/value list is used to represent examples for an induction program, the measurement of certain attributes and not others places limits on the kinds of patterns that the learner can find. The learner is said to be *biased* by its observation language. The hypothesis language also places constraints on what may and may not be learned. For example, in the language of attributes and values, relationships between objects are difficult to represent. Whereas, a more expressive language, such as first-order logic, can easily be used to describe relationships. These biases are collectively referred to as *representation bias*.

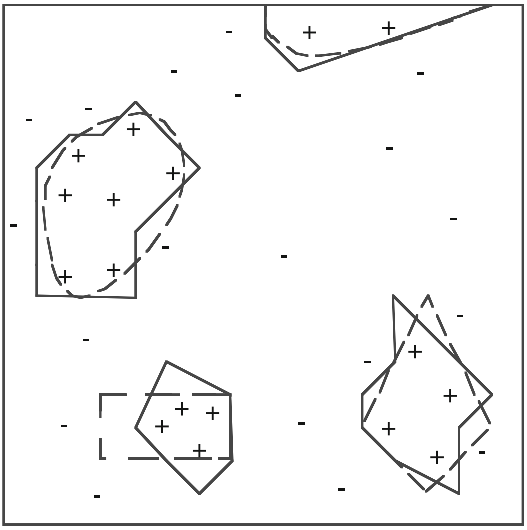
Representational power comes at a price. Learning can be viewed as a search through the space of all sentences in a language for a sentence that best describes the data. The richer the language, the larger the search space. When the search space is small, it is possible to use "brute force" search methods. If the search space is very large, additional knowledge is required to reduce the search. Notions of generality and specificity are important for ordering the search (see ▶ Generalization and ▶ Specialization).

Representation

The representation of instances and concepts affects the way a learning system searches for concept representations.

The input to a learning program may take many forms, for example, records in a database, pages of text, images, audio, and other signals of continuous data. Very often, the raw data are transformed into *feature vectors* or *attribute/value lists*. The values of the attributes or features may be continuous or discrete. These representation by attribute/value lists is the observation language.

The representation of the concept varies considerably, depending on the approach taken for learning. In ▶ instance-based learning, concepts are represented by a set of prototypical instances of the concept, so abstract representations are not constructed at all. This kind of representation is said to be *extensional*. Instance-based learning is also called ▶ lazy learning because the learner does little work at the time that training instances are presented. Rather, at classification time, the system must find the most similar instances to the new example. See Fig. 1.

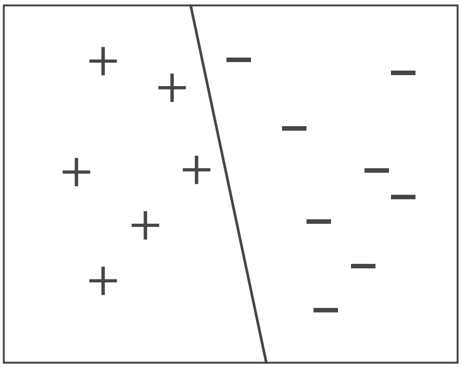


Learning as Search, Fig. 1 The extension of an instance-based Learning concept is shown in *solid lines*. The *dashed lines* represent the target concept. A sample of positive and negative examples is shown (Adapted from Aha et al. 1991)

When instances are represented as feature vectors, we can treat each feature or attribute as one dimension in a multi-dimensional space. The supervised learning problem can then be characterized as the problem of finding a surface that separates objects that belong to different classes into different regions. In the case of unsupervised learning, the problem becomes one of the finding clusters of instances in the multi-dimensional space.

Learning methods differ in the way they represent and create the discrimination surfaces. In *function approximation*, the learner searches for functions that describes the surface (Fig. 2). Function approximation methods can often produce accurate classifiers because they are capable of construction complex decision surfaces. However, the concept description is stored as a set of coefficients. Thus, the results of learning are not easily available for inspection by a human reader.

Rather than searching for discriminant functions, symbolic learning systems find expressions equivalent to sentences in some form of logic. For example, we may distinguish objects according to two attributes: size and color. We may say that an object belongs to class 3 if its color is red and its size is very small to medium. Following the notation of Michalski (1983), the classes in Fig. 3 may be written as:



Learning as Search, Fig. 2 A linear discrimination between two classes

$$\text{class1} \leftarrow \text{size} = \text{large} \wedge \text{color} \in \{\text{red, orange}\}$$

$$\text{class2} \leftarrow \text{size} \in \{\text{small, medium}\} \wedge \text{color} \in \{\text{orange, yellow}\}$$

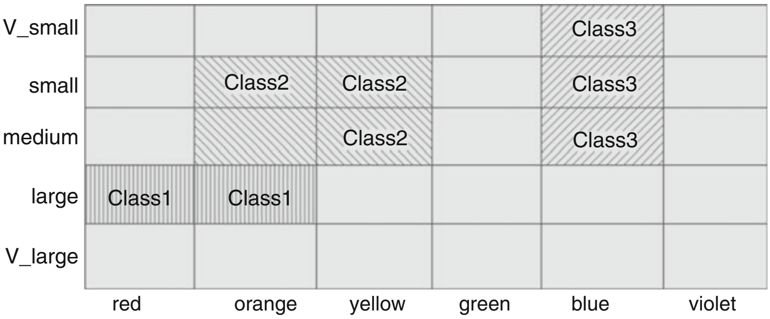
$$\text{class3} \leftarrow \text{size} \in \{v\_\text{small} \ldots \text{medium}\} \wedge \text{color} = \text{blue}$$

Note that this kind of description partitions the universe with axis-orthogonal surfaces, unlike the function approximation methods that find smooth surfaces to discriminate classes (Fig. 4).

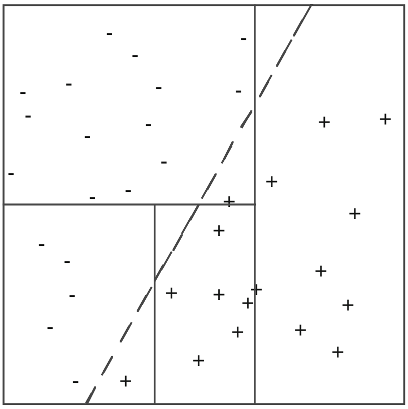
Useful insights into induction can be gained by visualizing it as searching for a discrimination surface in a multi-dimensional space. However, there are limits to this geometric interpretation of learning. If we wish to learn concepts that describe complex objects and relationships between the objects, it is often useful to rely on reasoning about the concept description language itself.

As we saw, the concepts in Fig. 3 can be expressed as clauses in propositional logic. We can establish a correspondence between sentences in the concept description language (the hypothesis language) and a diagrammatic representation of the concept. More importantly, we can create a correspondence between generalization and spe-

Learning as Search, Fig. 3 Discrimination on attributes and values



	red	orange	yellow	green	blue	violet
V_small					Class3	
small		Class2	Class2		Class3	
medium			Class2		Class3	
large	Class1	Class1				
V_large						



Learning as Search, Fig. 4 The *dashed line* shows the real division of objects in the universe. The *solid lines* show a decision tree approximation

cialization operations on the sets of objects and generalization and specialization operations on the sentences of the language.

Once we have established the correspondence between sets of objects and their descriptions, it is often convenient to forget about the objects and only consider that we are working with expressions in a language. For example, the clause

$$\text{class2} \leftarrow \text{size} = \text{large} \wedge \text{color} = \text{red} \qquad (1)$$

can be generalized to

$$\text{class} \leftarrow \text{size} = \text{large} \qquad (2)$$

by dropping one of the conditions. Thus, we can view learning as search through a generalization lattice that is created by applying different syntactic transformations on sentences in the hypothesis language.

Version Spaces and Subsumption

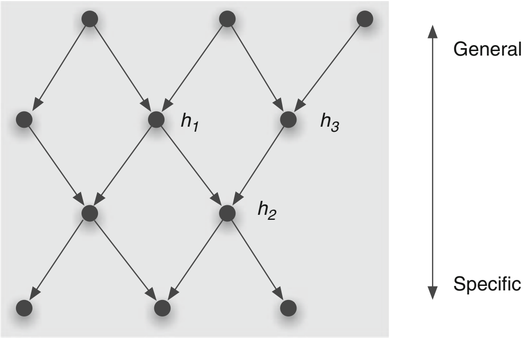
Mitchell (1977, 1982) defines the *version space* for a learning algorithm as the subset of hypotheses consistent with the training examples. That is, the hypothesis language is capable of describing a large, possibly infinite, number of concepts. When searching for the target concept, we are only interested in the subset of sentences in the hypothesis language that are consistent with the training examples, where consistent means that the examples are correctly classified. We can used the *generality* of concepts to help us limit our search to only those hypotheses in the version space.

In the above example, we stated that clause (2) is more general than clause (1). In doing so, we assumed that there is a general-to-specific ordering on the sentences in the hypothesis language. We can formalize the generality relation as follows. A hypothesis, h, is a predicate that maps an instance to *true* or *false*. That is, if $h(x)$ is true then x is hypothesized to belong to the concept being learned, the *target*. Hypothesis, h_1, is more general than or equal to h_2, if h_1 covers at least as many examples as h_2 (Mitchell 1997). That is, $h_1 \geq h_2$ if and only if

$$(\forall x)[h_1(x) \rightarrow h_2(x)]$$

A hypothesis, h_1, is strictly more general than h_2, if $h_1 \geq h_2$ and $h_2 \not\leq h_1$.

Note that the *more general than* ordering is strongly related to *subsumption* (see ▸ subsumption and the ▸ Logic of Generality). Where the above definition of the generality relation is given in terms of the cover of a



Learning as Search, Fig. 5 Generalization lattice

hypothesis, subsumption defines a generality ordering on expressions in the hypothesis language.

Learning algorithms can use the *more general than* relation to order their search for the best hypothesis. Because generalizations and specializations may not be unique, this relation forms a lattice over the sentences in the hypothesis language, as illustrated in Fig. 5. A search may start from the set of most specific hypotheses that fit the training data and perform a *specific-to-general* search or it may start from the set of most general hypotheses and perform a *general-to-specific* search. The search algorithm may also be bidirectional, combining both.

In Fig. 5, each node represents a hypothesis. The learning algorithm searches this lattice in an attempt to find the hypothesis that best fits the training data. Like searching in any domain, the algorithm may keep track of one node at a time, as in *depth first* or *best first* searches, or it may create a frontier of nodes as in *breadth first* or *beam searches*.

Suppose we have single-hypothesis search. A specific-to-general search may begin by randomly selecting a positive training example and creating a hypothesis that the target concept is exactly that example. Each time a new positive example is seen that is not covered by the hypothesis, the hypothesis must be *generalized*. That is, a new hypothesis is constructed that is general enough to cover all the examples covered by the previous hypothesis, as well as covering the new example. If the algorithm sees a negative example that is incorrectly covered by the current hypothesis, then the hypothesis must be *specialized*. That is, a new hypothesis is construct that is more specific than the current hypothesis such that all the positive examples that were previously covered are still covered by the new negative example is excluded.

A similar method can be used for a general-to-specific search. In this case, the initial hypothesis is that the target concept covers every object in the universe. In both cases, the algorithm must choose how to construct either generalizations or specializations. That is, a method is needed to choose which nodes in the search to expand next. Here, the ▸ least general generalization (Plotkin 1970) or the ▸ most general specialization are useful. These define the smallest steps that can be taken in expanding the search. For example, in Fig. 5, h_2 is the minimal specialization that can be made from h_1 or h_3 in a general-to-specific search that starts from the top of the lattice. Similarly, h_1 and h_3 are the least general generalizations of h_2. A search for the target concept can begin with an initial hypothesis and make minimal generalizations or specializations in expanding the next node in the search.

Rather than maintaining on a single current hypothesis, a search strategy may keep a set of candidate hypotheses. For example, a breadth first search generalizing from hypothesis h_2 will create a frontier for the search that is the set $\{h_1, h_3\}$. When there are many ways in which an hypothesis can be generalized or specialized, the size of the frontier set may be large. In algorithms such as Aq (Michalski 1983) and CN2 (Clark and Niblett 1989), a *beam search* is used. Rather than storing all possible hypotheses, the n best are kept are stored, where "best" can be defined in several ways. One metric for comparing hypotheses is given by

$$\frac{P_c + N_{\bar{c}}}{P + N}$$

where P and N are the number of positive and negative instances, respectively; P_c is the number of positive instances covered by the hypothesis;

and $N_{\bar{c}}$ is the number of negative instances not covered.

Algorithm 1 The candidate-elimination algorithm, after Mitchell (1997)

```
Initialize G to the set of maximally general hypotheses in the hypothesis space
Initialize S to the maximally specific hypotheses in the hypothesis space
For each training example, $d$,
  if $d$ is a positive example
    remove from $G$ any hypothesis inconsistent with $d$
    For each hypothesis, $s$, in S that is not consistent with $d$
      remove $s$ from $S$
        add all minimal generalizations, $h$, of s such that
          $h$ is consistent with $d$ and some member of $G$ is more general than $h$
remove from $S$ any hypothesis that is more general than another hypothesis in $S$
  if $d$ is a negative example
    remove from $S$ any hypothesis inconsistent with $d$
    For each hypothesis, $g$, in $G$ that is not consistent with $d$
      remove $g$ from $G$
        add all minimal specializations, $h$, of $g$ such that
          $h$ is consistent with $d$ and some member of $S$ is more general than $h$
      remove from $G$ any hypothesis that is less general than another hypothesis in $G$
```

Mitchell's (1997) candidate-elimination algorithm performs a bidirectional search in the hypothesis space. It maintains a set, S, of most specific hypotheses that are consistent with the training data and a set, G, of most general hypotheses consistent with the training data. These two sets form two boundaries on the version space. As new training examples are seen, the boundaries are generalized or specialized to maintain consistency. If a new positive example is not covered by a hypothesis in S, then it must be generalized. If a new negative example is not rejected by an hypotheses in G, then it must be specialized. Any hypothesis in G not consistent with a positive example is removed and any hypothesis in S not consistent with a negative example is also removed. See Algorithm 1.

Noisy Data

Up to this point, we have assumed that the training data are free of noise. That is, all the examples are correctly classified and all the attribute values are correct. Once we relax this assumption, the algorithms described above must be modified to use approximate measures of consistency. The danger presented by noisy data is that the learning algorithm will *over fit* the training data by creating concept descriptions that try to cover the bad data as well as the good. For methods to handle noisy data see the entries in ▶ pruning.

Several standard texts give good introductions to search in learning, including Langley (1996), Mitchell (1997), Bratko (2000), and Russell and Norvig (2009).

Cross-References

▶ Decision Tree
▶ Generalization
▶ Induction
▶ Instance-Based Learning
▶ Logic of Generality
▶ Rule Learning
▶ Subsumption

Recommended Reading

Aha DW, Kibler D, Albert MK (1991) Instance-based learning algorithms. Mach Learn 6(1):37–66

Bratko I (2000) Prolog programming for artificial intelligence, 3rd edn. Addison-Wesley, Boston

Clark P, Niblett T (1989) The CN2 induction algorithm. Mach Learn 3(4):261–283

Langley P (1996) Elements of machine learning. Morgan Kaufmann, San Mateo

Michalski RS (1983) A theory and methodology of inductive learning. In R. S

Michalski RS, Carbonell JG, Mitchell TM (eds) (1983) Machine learning: an artificial intelligence approach. Tioga, Palo Alto

Mitchell TM (1977) Version spaces: a candidate elimination approach to rule-learning. In: Proceedings of the fifth international joint conference on artificial intelligence, Cambridge, pp 305–310

Mitchell TM (1982) Generalization as search. Artif Intell 18(2):203–226

Mitchell TM (1997) Machine learning. McGraw-Hill, New York

Plotkin GD (1970) A note on inductive generalization. In: Meltzer B, Michie D (eds) Machine intelligence, vol 5, pp 153–163. Edinburgh University Press, Edinburgh

Russell S, Norvig P (2009) Artificial intelligence: a modern approach, 3rd edn. Prentice Hall, Englewood cliffs

Learning Bayesian Networks

▶ Learning Graphical Models

Learning Bias

▶ Inductive Bias

Learning by Demonstration

▶ Behavioral Cloning

Learning by Imitation

▶ Behavioral Cloning

Learning Classifier Systems

▶ Classifier Systems

Learning Control

Learning control refers to the process of acquiring a control strategy for a particular control system and a particular task by trial and error. Learning control is usually distinguished from adaptive control in that the learning system is permitted to fail during the process of learning. In contrast, adaptive control emphasizes single trial convergence without failure. Thus, learning control resembles the way that humans and animals acquire new movement strategies, while adaptive control is a special case of learning control that fulfills stringent performance constraints, e.g., as needed in life-critical systems like airplanes and industrial robots. In general, the control system can be any system that changes its state in response to a control signal, e.g., a web page with a hyperlink, a car, or a robot.

Learning Control Rules

▶ Behavioral Cloning

Learning Curves in Machine Learning

Claudia Perlich
IBM T.J. Watson Research Center, Yorktown Heights, NY, USA

Synonyms

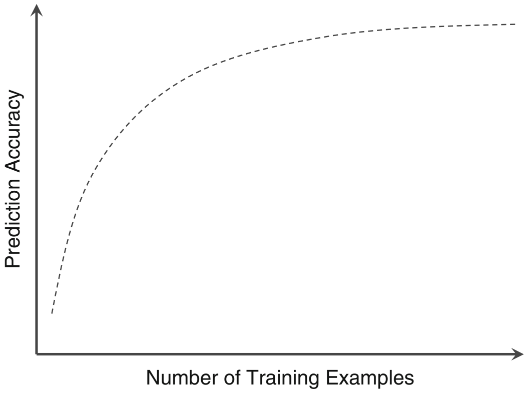
Error curve; Experience curve; Improvement curve; Training curve

Definition

A learning curve shows a measure of predictive performance on a given domain as a function of some measure of varying amounts of learning effort. The most common form of learning curves in the general field of machine learning shows predictive accuracy on the test examples as a function of the number of training examples as in Fig. 1.

Motivation and Background

Learning curves were initially introduced in educational and behavioral/cognitive psychology. The first person to describe the learning curve was Hermann Ebbinghaus in 1885 (Wozniak



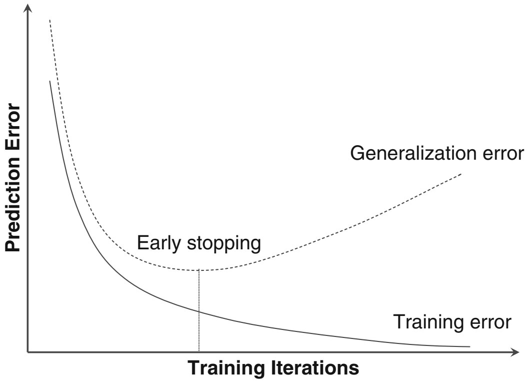
Learning Curves in Machine Learning, Fig. 1 Stylized learning curve showing the model accuracy on test examples as a function of the number of training examples

1999). He found that the time required to memorize a nonsense syllable increased sharply as the number of syllables increased. Wright (1936) described the effect of learning on labor productivity in the aircraft industry and proposed a mathematical model of the learning curve. Over time, the term has acquired related interpretation in many different fields including the above definition in machine learning and statistics.

Use of Learning Curves in Machine Learning

In the area of machine learning, the term "learning curve" is used in two different contexts, the main difference being the variable on the x-axis of the curve.

- The ▶ artificial neural network (ANN) literature has used the term to show the diverging behavior of in and out-of-sample performance as a function of the *number of training iterations* for a given number of training examples. Figure 2 shows this stylized effect.
- General machine learning uses learning curves to show the predictive ▶ generalization performance as a function of the *number of training examples*. Both the graphs in Fig. 3 are examples of such learning curves.



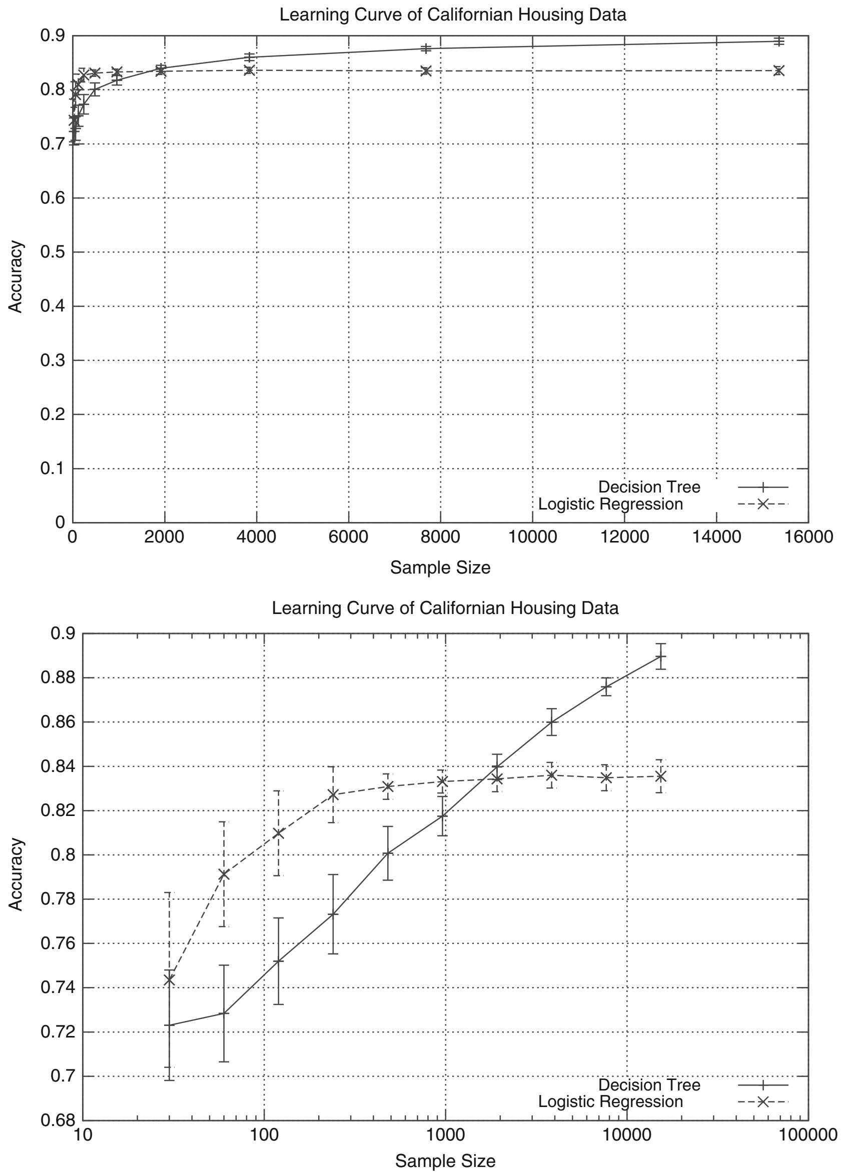
Learning Curves in Machine Learning, Fig. 2 Learning curve for an artificial neural network

Artificial Neural Networks

The origins of ANNs are heavily inspired by the social sciences and the goal of recreating the learning behavior of the brain. The original model of the "perceptron" mirrored closely the biological foundations of neural sciences. It is likely that the notion of learning curves was to some extent carried over from the social sciences of human learning into the field of ANNs. It shows the model error as a function of the training time measured in terms of the number of iterations. One iteration denotes in the context of neural network learning one single pass over the training data and the corresponding update of the network parameters (also called weights). The algorithm uses gradient descent minimizing the model error on the training data.

The learning curve in Fig. 2 shows the stylized effect of the relative training and generalization error on a test set as a function of the number of iterations. After initial decrease of both types of error, the generalization error reaches a minimum and starts to increase again while the training error continues to decrease.

This effect of increasing generalization error is closely related to the more general machine learning issue of ▶ overfitting and variance error for models with high expressive power (or capacity). One of the initial solutions to this problem for neural networks was early stopping – some form of early regularization technique that picked the model at the minimum of the error curve on a



Learning Curves in Machine Learning, Fig. 3 Typical learning curves in original and log scale

validation subset of the data that was not used for training.

General Machine Learning

In the more general machine learning setting and statistics (Flury and Schmid 1994), learning curves represent the generalization performance of the model as a function of the size of the training set.

Figure 3 was taken from Perlich et al. (2003) and shows two typical learning curves for two different modeling algorithms (▶ decision tree and ▶ logistic regression) on a fairly large domain. For smaller training-set sizes the curves are steep, but the increase in accuracy lessens for larger training-set sizes. Often for very large training-set sizes the standard representation in the upper graph obscures small, but non-trivial, gains. Therefore, to visualize the curves it is often useful to use a log scale on the horizontal axis and start the graph at the accuracy of the smallest training-set size (rather than at zero). In addition, one can include error bars that capture the estimated variance of the error over multiple experiments and provide some impression of the relevance of the differences between two learning curves as shown in the graphs.

The figure also highlights a very important issue in comparative analysis of different modeling techniques: learning curves for the same domain and different models can cross. This implies an important pitfall as pointed out by Kibler and Langley (1988): "Typical empirical papers report results on training sets of fixed size, which tells one nothing about how the methods would fare given more or less data, rather than collecting learning curves ⋯". A corollary on the above observation is the dangers of selecting an algorithm on a smaller subset of the ultimately available training data either in the context of a proof of concept pre-study or some form of cross-validation.

Aside from its empirical relevance there has been significant theoretical work on learning curves – notably by Cortes et al. (1994). They are addressing the question of predicting the expected generalization error from the training error of a model. Their analysis provides many additional insights about the generalization performance of different models as a function of not only training size but in addition the model capacity.

Cross-References

▶ Artificial Neural Networks
▶ Decision Tree
▶ Generalization Performance
▶ Logistic Regression
▶ Overfitting

Recommended Reading

Cortes C, Jackel LD, Solla SA, Vapnik V, Denker JS (1994) Learning curves: asymptotic values and rate of convergence. Adv Neural Inf Process Syst 6:327–334

Flury BW, Schmid MJ (1994) Error rates in quadratic discrimination with constraints on the covariance matrices. J Classif 11:101–120

Kibler D, Langley P (1988) Machine learning as an experimental science. In: Proceedings of the third European working session on learning, Pittman, Glasgow. Kluwer Academic, Hingham, pp 81–92

Perlich C, Provost F, Simonoff J (2003) Tree induction vs. logistic regression: a learning-curve analysis. J Mach Learn Res 4:211–255

Shavlik JW, Mooney RJ, Towell GG (1991) Symbolic and neural learning algorithms: an experimental comparison. Mach Learn 6:11–143

Wozniak RH (1999) Introduction to memory: Hermann Ebbinghaus (1885/1913). In: Classics in the history of psychology. Thoemmes Press, Bristol

Wright TP (1936) Factors affecting the cost of airplanes. J Aeronaut Sci 3(4):122–128

Learning from Complex Data

▶ Learning from Structured Data

Learning from Labeled and Unlabeled Data

▶ Semi-supervised Learning
▶ Semi-supervised Text Processing

Learning from Non-Propositional Data

▶ Learning from Structured Data

Learning from Nonvectorial Data

▶ Learning from Structured Data

Learning from Preferences

▶ Preference Learning

Learning from Structured Data

Tamás Horváth and Stefan Wrobel
Fraunhofer IAIS, Schloss Birlinghoven,
University of Bonn, Sankt Augustin, Germany

Synonyms

Learning from complex data; Learning from non-propositional data; Learning from nonvectorial data

Definition

Learning from structured data refers to all those learning tasks where the objects to be considered as inputs and/or outputs can usefully be thought of as possessing internal structure and/or as being interrelated and dependent on each other, thus forming a structured space. Typical instances of data in structured learning tasks are sequences as they arise, e.g., in speech processing or bioinformatics, and trees or general graphs such as syntax trees in natural language processing and document analysis, molecule graphs in chemistry, relationship networks in social analysis, and link graphs in the World Wide Web. Learning from structured data presents special challenges, since the commonly used feature vector representation and/or the i.i.d. (independently and identically distributed data) assumption are no longer applicable. Different flavors of learning from structured data are represented by (overlapping) areas such as ▶ Inductive Logic Programming, ▶ Statistical Relational Learning, probabilistic relational and logical learning, learning with structured outputs, sequence learning, learning with trees and graphs, ▶ graph mining, and ▶ collective classification.

Motivation and Background

For a long time, learning algorithms had almost exclusively considered data represented in rectangular tables defined by a fixed set of columns and a number of rows corresponding to the number of objects to be described. In this representation, each row independently and completely describes one object, each column containing the value of one particular property or feature of the object. Correspondingly, this representation is also known as feature vector representation, propositional representation, or vectorial data representation. Statistically, in such a representation, the values in each row (i.e., the objects) are assumed to be drawn i.i.d. from a fixed (but unknown) distribution.

However, when working with objects that are interrelated and/or have internal structure, this representation is no longer adequate. Consider representing chemical molecules with varying numbers of atoms and bonds in a table with a fixed number of columns. If we wanted each molecule to correspond to one row, we would have to fit the atoms and bonds into the columns, e.g., by reserving a certain number of columns for each one of them and their respective properties. To do that however, we would have to make the table wide enough to contain the largest possible molecule, resulting in many empty columns for smaller molecules, and by mapping the component atoms and bonds to columns, we would assign an order to them that would not be justified by the underlying problem and that would consequently mislead any feature vector learning algorithm.

The second issue with structured data arises from objects that are interrelated. Consider, e.g., the task of speech recognition, i.e., learning to map an acoustic unit into the corresponding lexical unit. Clearly, to solve this task, one must consider the sequence of such units, since both on the input and the output sides the probability of observing a particular unit will strongly depend on the preceding or subsequent units. The same is true, e.g., in classifying pages in the World Wide Web, where it is quite likely that the classification of the page will correlate with the classifications of neighboring pages. Therefore, any learning algorithm that would regard acoustic units or pages as independent and identically distributed objects is destined to fail, since for a successful solution the interdependencies must be modeled and exploited.

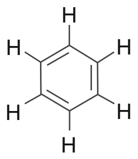
In machine learning, even though there has been interest in structured representation from the very beginning of the 1970s (cf. the systems Arch (Winston 1975) or INDUCE (Michalski 1983)), it was only in the 1990s, triggered by the popularity of logic programming and Horn clause representation, that learning from structured data was more intensively considered for logical representations in the subfield of Inductive Logic Programming. Outside of (what was then) machine learning, due to important applications such as speech processing, probabilistic models for sequence data such as ▸ Hidden Markov Models have been considered much earlier. Toward the end of the 1990s, given an enormous surge of interest in applications in bioinformatics and the World Wide Web, and technical advances resulting from the integration of probabilistic and statistical approaches into machine learning (e.g., ▸ Graphical Models and ▸ kernel methods), work on learning from structured data has taken off and now represents a significant part of machine learning research in overlapping subareas such as Inductive Logic Programming, Statistical Relational Learning, probabilistic relational and logical learning, learning with structured outputs, sequence learning, learning with trees and graphs, graph mining, and collective inference.

Main Tasks and Solution Approaches

A particular problem setting for learning from structured data is given by specifying, among others, (1) the language representing the input and output of the learning algorithms, (2) the type of the input and/or output data, and (3) the learning task.

1. Beyond attribute-value representation, the most intensively investigated representation languages used in learning are ▸ First-Order Logic, in particular, the fragment of first-order Horn clauses, and labeled graphs. Although labeled graphs can be considered as special relational structures and thus form a special fragment of first-order logic, these two representation languages are handled separately in machine learning. As an example of first-order representation of labeled graphs, the molecular graph of a benzene ring can be represented as follows:

   atom(a_1,carbon).,$\ldots$,atom(a_6,carbon).,
   atom(a_7,hydrogen).,$\ldots$,atom(a_{12},hydrogen).,
   edge(a_1,a_2,aromatic).,$\ldots$,edge(a_6,a_1,aromatic).,
   edge(a_1,a_7,single).,$\ldots$,edge(a_6,a_{12},single).,
   edge(X,Y) $\leftarrow$ edge(Y,X).

   The molecular graph of benzene rings (carbon atoms are unmarked)

   Besides complexity reasons, the above two representation languages are motivated also by the difference in the matching operators typically used for these two representations. While in case of first-order logic, the matching operator is defined by logical implication or by relational homomorphism (often referred to as subsumption), which is a decidable, but thus, incomplete variant of logical implication, in case of labeled graphs it is defined by subgraph isomorphism (i.e., by injective homomorphism).
2. Another component defining a task for learning from structured data is the type of the input and/or output data (see ▸ Observation Language and ▸ Hypothesis Language). For the input, two main types can be distinguished: the instances are disjoint structures (structured instances) or substructures of some global structure (structured instance space). Molecular graphs formed by the atom-bond structure of chemical compounds are a common example of structured instances. For structured instance spaces, the web graph provides an example of a global structure; for this case, the set of instances corresponds to the set of vertices formed by the web sites. The primary goal of traditional discriminative learning

is to approximate unknown target functions mapping the underlying instance space to some subset of the set of real numbers. In some of the applications, however, the elements of the range of the target function must also be structured. Such problems are referred to as learning in structured output spaces. As an example of structured output, we mention the protein secondary structure prediction problem, where the goal is to approximate the function mapping the primary structures of proteins to their secondary structures. Notice that primary and secondary structures can be represented by strings, which in turn can be considered as labeled directed paths.

3. Finally, the third component defining a problem setting is the learning task. Besides the classical learning tasks (e.g., supervised, semisupervised, unsupervised, transductive learning etc.), recent tasks include new problems such as, e.g., learning preferences (i.e., a directed graph, where an edge from vertex u to vertex v denotes that v is preferred to u), learning rankings (i.e., when the target preference relation must be a total order), etc.

Several classes of algorithms have been developed for the problem settings defined by the above components. ▸ Propositionalization techniques (e.g., as in LINUS (Lavrac et al. 1991)) first transform the structured data into a single table of fixed width by extracting a large number of propositional features and then use some propositional learner.

Non-propositionalization rule-based approaches follow mainly general-to-specific (top–down) or specific-to-general (bottom-up) search strategies. For top–down search (e.g., as in FOIL (Quinlan 1990)), the crucial step of the algorithms is the definition of the refinement operators. While for graph structured data the specialization relation on the hypothesis space is usually defined by subgraph isomorphism and is therefore a partial order, for First-Order Logic it is typically defined by subsumption and is therefore only a preorder (i.e., antisymmetry does not hold), leading to undesirable algorithmic properties (e.g., incompleteness). For bottom–up search (e.g., as in GOLEM (Muggleton and Feng 1992)), which is less common for graph structured data, the generalization of hypotheses is usually defined by some variant of Plotkin's Least General Generalization operator for first-order clauses. While this generalization operator has nice algebraic properties, its application raises severe complexity issues, as the size of the hypotheses may exponentially grow in the number of examples.

Recent research in structural learning has been focusing very strongly on distance- and kernel-based approaches which in terms of accuracy have often turned out superior to rule-based approaches (e.g., in virtual screening of molecules). In such approaches, the basic algorithms carry over unchanged from the propositional case; instead, special distance (e.g., as in RIBL (Emde and Wettschereck 1996)) or kernel functions for structural data are developed. Since even for graphs, computing any complete kernel (i.e., for which the underlying embedding function into the feature space is injective) is at least as hard as the graph isomorphism problem, most practical and efficient kernels are based on examining the structure for the occurrence of simpler parts (e.g., trees, walks, paths, and cycles) which are then counted and effectively used as feature vectors in an intersection kernel.

Finally, as a recent class of approaches, we also mention Statistical Relational Learning which extends probabilistic Graphical Models (e.g., Bayesian networks or Markov networks) with relational and logic elements (e.g., as in Alchemy (Domingos and Richardson 2007), ICL (Poole 2008), PRISM (Sato and Kameya 2008)).

Applications

Virtual compound screening is a representative application example of learning from structured data. This computational problem in pharmaceutical research is concerned with the identification of chemical compounds that can be developed into drug candidates. Since current pharmaceutical compound repositories contain millions of molecules, the design of efficient algorithms for virtual compound screening has become an integral part of computer-aided drug design. One of the branches of the learning algorithms concerned with this prediction problem is based on using the compounds' 2D graph structures formed by their atoms and bonds. Depending on the representation of chemical graphs, this branch of algorithms can further be classified into logic and graph-based approaches. The first class of algorithms, developed mostly in Inductive Logic Programming, treats chemical graphs as relational structures addressing the problem to the context of learning in logic; the second class of algorithms regards them as labeled graphs addressing the problem to ▶ Graph Mining.

Cross-References

▶ Hypothesis Language
▶ Inductive Logic Programming
▶ Observation Language
▶ Statistical Relational Learning
▶ Structured Induction

Recommended Reading

Cook D, Holder L (eds) (2007) Mining graph data. Wiley, New York

De Raedt L (2008) From inductive logic programming to multi-relational data mining. Springer, Heidelberg

Domingos P, Richardson M (2007) Markov logic: a unifying framework for statistical relational learning. In Getoor L, Taskar B (eds) Introduction to statistical relational learning. MIT Press, Cambridge, MA, pp 339–371

Emde W, Wettschereck D (1996) Relational instance based learning. In: Saitta L (ed) Proceedings of the 13th international conference on machine learning. Morgan Kaufmann, San Francisco, pp 122–130

Gärtner T (2003) A survey of kernels for structured data. SIGKDD Explor 5(1):49–58

Getoor L, Taskar B (eds) (2007) Introduction to relational statistical learning. MIT Press, Cambridge, MA

Lavrac N, Dzeroski S, Grobelnik M (1991) Learning nonrecursive definitions of relations with LINUS. In Kodratoff Y (ed) Proceedings of the 5th European working session on learning. Lecture notes in computer science, vol 482. Springer, Berlin, pp 265–281

Michalski RS (1983) A theory and methodology of inductive learning. In Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning: an artificial intelligence approach. Morgan Kaufmann, San Francisco, pp 83–134

Muggleton SH, De Raedt L (1994) Inductive logic programming: theory and methods. J Logic Program 19,20:629–679

Muggleton SH, Feng C (1992) Efficient induction of logic programs. In: Muggleton S (ed) Inductive logic programming. Academic Press, London, pp 291–298

Poole D (2008) The independent choice logic and beyond. In: De Raedt L, Frasconi P, Kersting K, Muggleton S (eds) Probabilistic inductive logic programming: theory and application. Lecture notes in artificial intelligence, vol 4911. Springer, Berlin

Quinlan JR (1990) Learning logical definitions from relations. Mach Learn 5(3):239–266

Sato T, Kameya Y (2008) New advances in logic-based probabilistic modeling by PRISM. In De Raedt L, Frasconi P, Kersting K, Muggleton S (eds) Probabilistic inductive logic programming: theory and application. Lecture notes in artificial intelligence, vol 4911. Springer, Berlin, pp 118–155

Winston PH (1975) Learning structural descriptions from examples. In: Winston PH (ed) The psychology of computer vision. McGraw-Hill, New York, pp 157–209

Learning Graphical Models

Kevin B. Korb
Clayton School of Information Technology,
Monash University, Clayton, VIC, Australia

Abstract

Learning graphical models has become an important part of data mining and data science. Here we survey some of the more important techniques and concepts, including causal models and causal discovery, statistical equivalence, Markov Blanket discovery and knowledge engineering.

Synonyms

Bayesian model averaging; Causal discovery; Dynamic Bayesian network; Learning Bayesian networks

Definition

Learning graphical models (see Graphical Models) means to learn a graphical representation of either a causal or probabilistic model containing the variables $X_j \in \{X_i\}$. Although graphical models include more than directed acyclic graphs (DAGs), we shall focus here on learning DAGs, as that is where the majority of research and application is taking place.

Definition 1 (Directed acyclic graph (DAG)) A directed acyclic graph (DAG) is a set of variables (nodes, vertices) $\{X_i\}$ and a set of directed arcs (edges) between them such that following the arcs in their given direction can never lead from a variable back to itself.

DAGs parameterized to represent probability distributions are otherwise known as *Bayesian networks*. Some necessary concepts and notation for discussing the learning of graphical models is given in Table 1.

A key characteristic of multivariate probability distributions is the conditional independence structure they give rise to, i.e., the complete list of statements of the form

$$X_A \perp\!\!\!\perp X_B | X_C$$

true of the distribution. The goal of learning DAGs is to learn a minimal DAG representation of the conditional independence structure for a distribution satisfying the Markov condition:

Definition 2 (Markov condition) A DAG satisfies the Markov condition relative to a probability distribution if and only if for all X_i and $X_j \notin \pi_{X_i}$ in the DAG $X_i \perp\!\!\!\perp X_j | \pi_{X_i}$ so long as X_j is not a descendant of X_i (i.e., X_j is not in the transitive closure of the parent relation starting from X_i).

DAGs which violate the Markov condition are not capable of fully representing the relevant probability distribution. Upon discovering such a violation, the normal response is to fix the model by adding missing arcs. In the causal discovery literature, this condition is often referred to as the *causal Markov condition*, which simply means the arcs are being interpreted as representing causal relationships and not merely as probabilistic dependencies.

Definition 3 (Markov blanket) The Markov blanket (MB) of a node X_i is the minimal set $\mathbf{X_{MB}}$ such that for all other nodes X_j in the model $X_i \perp\!\!\!\perp X_j | \mathbf{X_{MB}}$.

The Markov blanket consists of a node's parents, children, and its children's other parents.

Motivation and Background

Bayesian networks have enjoyed a substantial success in thousands of diverse modeling, prediction, and control applications, including medical diagnosis, epidemiology, software engineering, ecology and environmental management, agriculture, intelligence and security, finance, and

Learning Graphical Models, Table 1 Notation

Notation	Description
X_i	A random variable
$\mathbf{X}$	A set of random variables
$\{X_i\}$	A set of random variables indexed by $i \in I$
$X = x_j$ (or, x_j)	A random variable taking the value x_j
$p(x)$	The probability that $X = x$
$X_A \models X_B$	X_A and X_B are *independent* (i.e., $p(X_A) = p(X_A \vert X_B)$)
$X_A \models X_B \vert X_C$	X_A and X_B are *conditionally independent* given X_C (i.e., $p(X_A \vert X_C) = p(X_A \vert X_B, X_C)$)
$X_A \not\models X_B$	X_A and X_B are *dependent* (i.e., $p(X_A) \neq p(X_A \vert X_B)$)
$X_A \not\models X_B \vert X_C$	X_A and X_B are *conditionally dependent* given X_C (i.e., $p(X_A \vert X_C) \neq p(X_A \vert X_B, X_C)$)
π_{X_i}	The set of parents of X_i in a DAG (i.e., nodes Y such that $Y \rightarrow X_i$)

marketing (see, e.g., http://www.norsys.com for customers implementing such applications and more). Many of these networks have been built by the traditional process of "knowledge engineering," that is, by eliciting both structure and conditional probabilities from human domain experts. That process is limited by the availability of expertise and also by the time and cost of performing such elicitation and subsequent model validation. In domains where significant quantities of data are available, it is pertinent to consider whether automated learning of Bayesian networks might either replace or compliment knowledge engineering. A variety of techniques for doing so have been developed, and the causal discovery of Bayesian networks is by now an important subindustry of data mining.

Theory

Probability and Causality

The key to learning Bayesian networks from the sample data is the relation between causal dependence and probabilistic dependence. This is most easily understood in reference to undirected chains of variables, as in Fig. 1.

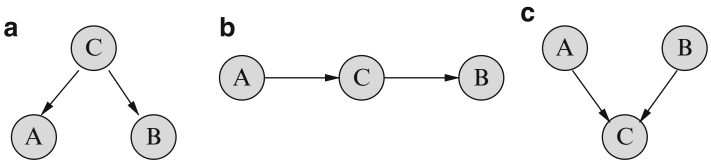
Where the arcs in Fig. 1 represent causal dependencies, then the probabilistic dependencies are as the caption describes, that is, in common causes and chains, the end nodes A and B are rendered probabilistically independent of each other given the knowledge of the state of C. Contrariwise, when A and B are parents of a common effect, and otherwise unrelated, they are probabilistically independent given no information (i.e., marginally independent), but *become* dependent given the knowledge of C. This last relationship is often called "explaining away," because it corresponds to situations where, when already knowing the presence of, say, some symptom C, the learning of the presence of a disease A reduces our belief in some alternative explanation B of the symptom.

These relationships between probabilistic dependence and causal dependence are the key for learning the causal structure of Bayesian networks because the sample data allow us to estimate probabilistic dependencies directly, and the difference between conditional dependency structures in Fig. 1a, b versus its opposite in Fig. 1c allows automated learners to distinguish between these different underlying causal patterns. (This is related to *d-separation* in graphical models.) This distinction is explicitly made use of in constraint learners, but also implicitly used by metric learners.

In addition to structure learning, parameter learning is necessary, i.e., learning the conditional probabilities of nodes given their parent values (conditional probability tables). Straightforward counting methods are frequently employed, although expectation maximization, Gibbs sampling, and other techniques may come into play when the available data are noisy.

Statistical Equivalence

Two DAGs are said to be statistically equivalent (or Markov equivalent) when they contain the same variables, and each can be parameterized so as to represent any probability distribution that the other can represent. Verma and Pearl (1990) proved that DAGs are statistically equivalent just in case they have the same undirected arc structures and the identical set of uncovered common



Learning Graphical Models, Fig. 1 Causality and probabilistic dependence: (**a**) common cause with $A \perp\!\!\!\perp B|C$; (**b**) causal chain with $A \perp\!\!\!\perp B|C$; (**c**) common effect with $A \not\perp\!\!\!\perp B|C$

effects, i.e., common effects such as in Fig. 1c where the two parents are not themselves directly connected. They dubbed the set of statistically equivalent models *patterns*; these can be represented using partially directed acyclic graphs (PDAGs), i.e., graphs with some arcs left undirected. Chickering (1995) showed that statistically equivalent models have identical maximum likelihoods relative to any given set of data. This latter result has suggested to many that causal learning programs can have no reasonable ambition to learn anything other than patterns, that is, any learner's discrimination between DAGs within a common pattern can only be based upon prior probability (e.g., prejudice). This is suggested, for example, by the fact that Bayesian learning combines (by multiplying) prior probabilities and likelihoods, so identical likelihoods will always lead to identical conclusions should the priors also be the same. We shall note some reason to doubt this supposed limit to causal discovery below.

Applications

Constraint Learners

The most direct application of the above ideas to learning Bayesian networks is exhibited in what may be called constraint learners. These programs assess conditional independencies between paired sets of variables given some other set of observed variables using statistical tests on the data, eliminating all DAGs that are incompatible with the independencies and dependencies asserted by the statistical test. (For this reason these programs are often called "conditional independence learners"; however that tag is misleading, as is explained below.) The original such algorithm, the IC algorithm of Verma and Pearl (1990), can be described in simplified form as three rules for constructing a network from Yes or No answers to questions of the form "Is it the case that $X \perp\!\!\!\perp Y \mid \mathbf{W}$?"

Rule I: Put an undirected link between any two variables X and Y if and only if for every set of variables $\mathbf{W}$ s.t. $X, Y \notin \mathbf{W}$

$$X \not\perp\!\!\!\perp Y \mid \mathbf{W},$$

> i.e., X and Y are directly connected if and only if they are dependent under every conditioning set (including Ø).

Rule II: For every undirected structure $X - Y - Z$, orient the arcs $X \rightarrow Y \leftarrow Z$ if and only if

$$X \not\perp\!\!\!\perp Z \mid \mathbf{W}$$

for **every W** s.t. $X, Z \notin \mathbf{W}$ and $Y \in \mathbf{Z}$,

> i.e., Y is an uncovered common effect if and only if the end variables X and Z are dependent under every conditioning set that includes Y.

Rule I is justified by the need to express the probabilistic dependency between X and Y under all possible circumstances. Rule II is justified by the asymmetry in probabilistic dependencies illustrated in Fig. 1.

Application of these two rules is then followed by applying a Rule III, which just checks for any arc directions that are forced by further considerations, such as avoiding the introduction of cycles or any uncovered common effects not already identified in Rule II, and so not supported by the conditional independence tests.

This algorithm was first put into practice in the PC algorithm distributed as a part of the TETRAD program (Spirtes et al. 1993). Aside from introducing some algorithmic efficiencies, PC adds orthodox statistical tests to answer the conditional independence questions. In the case of linear models, it uses a statistical significance test for vanishing partial correlations, accepting a dependence when and only when the test is statistically significant. For discrete networks a χ^2 test replaces the correlation test. Margaritis and Thrun further improve the algorithm's efficiency by limiting conditioning sets to the Markov blankets of the variable under test (Margaritis and Thrun 2000). The PC algorithm has become the most widely used Bayesian network learner, available in `weka` and many Bayesian network modeling tools.

Metric Learners

Constraint learners attempt to build up a network using a sequence of independent statistical tests. One problem with them is that when one such test gives an incorrect result, subsequent tests will assume that result, with the potential for errors to cascade. Metric learners, by contrast, use some score applied to a network as a whole to assess it relative to the data. The earliest of this kind, by Cooper and Herskovits, turned the computation of a Bayesian measure into a counting problem. Under a number of fairly strong assumptions, such as that children variable states are always uniformly distributed given their parent states, they derived the measure

$$P(d,e) = P(d)\prod_{k=1}^{n}\prod_{j=1}^{s_{\pi(k)}}\frac{(s_k-1)!}{(S_{kj}+s_k-1)!}\prod_{l=1}^{s_k}\alpha_{kjl}!$$

where d is the DAG being scored, e the data, n the number of variables, s_k the number of values X_k may take, $s_{\pi(k)}$ the number of values the parents of X_k may take, S_{kj} the number of cases in the data where π_k takes its j-th value, and α_{kjl} is the number of cases where X_k takes its l-th value and π_k takes its j-th value. Cooper and Herskovits proved that this measure can be computed in polynomial time. Assuming the adequacy of this probability distribution, computation of the joint probability suffices for Bayesian learning, since by Bayes' theorem, maximizing $P(d,e)$ is equivalent to maximizing the posterior probability of d. Cooper and Herskovits applied this measure in the program K2, which required as inputs both the data and a total ordering of the variables. The latter input eliminates all problems about discovering arc orientations, which could be considered a cheat since, as the discussion of the IC algorithm showed, this is a part of the causal learning problem. Subsequently, Chow and Liu's (1968) maximum-weighted spanning tree algorithm (MWST) has been used as a preprocessor to K2, doing a reasonable job of finding an ordering based upon the mutual information between pairs of variables.

A wide variety of alternative metrics for DAGs have been developed since K2. Heckerman et al. (1994) generalized the K2 metric to incorporate prior information, yielding BD (Bayesian metric with Dirichlet priors). Other alternatives include minimum description length (MDL) scores (Bouckaert 1993; Suzuki 1996, 1999), Bayesian information criterion (BIC) (Cruz-Ramírez et al. 2006), and minimum message length (MML) (Wallace et al. 1996; Korb and Nicholson 2011). Although all of these measures score the DAG as a whole relative to some data set, they are just as (or more) sensitive to the individual dependencies and independencies between variables as are the constraint learners. The difference between the two types of learners is not whether they attend to the sets of conditional independencies expressed in the data, but whether do so serially (which the constraint learners do) or collectively (as do the metric learners).

The question naturally arises whether constraint learners as a class are superior to metric learners or vice versa or, indeed, in which individual learner might be best. There is no settled answer to such questions, nor, in fact, is there any agreement about *how* such questions are best settled, even for fixed domains or data sets. Perhaps the issue is more general than that of learning Bayesian networks, since the fundamental theory of machine learning evaluation seems to be massively underdeveloped (see Algorithm Evaluation). In consequence, while nearly every new publication claims superiority in some sense for its preferred algorithm, the evidential basis for such claims remains a suspect. It is clear nonetheless that many of the programs available are helpful with data analysis and are being so applied.

Search and Complexity

The space of DAGs is superexponential in the number of variables, making the learning process hard; it is NP-hard to be exact (Chickering et al. 2004). In practice there are limits to the effectiveness of each algorithm, imposed by the number of variables (see Dimensionality Reduction), the number of joint states the variables may take, and the amount of data. The known limitations for different algorithms are scattered throughout the literature. This and the next section introduce some ideas for scaling up causal discovery.

Greedy search has frequently been used with both constraint-based and metric-based learning. The PC algorithm, searching the space of patterns, is an example, as it starts with a fully connected graph and searches greedily for arcs to remove. Chickering and Meek's greedy equivalence search (GES) is another greedy algorithm operating in the pattern space (Chickering and Meek 2002). Cooper and Herskovits' K2 is also a greedy searcher, adding arcs so long as single arc additions increase the probability score for the network. Bouckaert adopted this approach with his MDL score (Bouckaert 1993). Greedy searches, of course, tend to get lost in local maxima, and Suzuki loosened the search method for his MDL scoring, using branch and bound (Suzuki 1999).

Genetic algorithms (GAs) have been successfully applied to learning Bayesian networks. Larrañaga et al. used GAs over the space of total orderings to maximize the K2 score (Larrañaga et al. 1996); Neil and Korb developed a GA searching the DAG space to maximize the MML score (Neil and Korb 1999). A similar approach using MDL is found in Wong et al. (1999).

Markov chain Monte Carlo (MCMC) searches perform stochastic sampling over the model space and have become a popular technique for Bayesian network learning. Gibbs sampling is used in Chickering and Heckerman (1997), where they compare a number of different metrics (and incorrectly conflate BIC and MDL scores; see Cruz-Ramírez et al. 2006) for learning a restricted class of Bayesian networks. Another MCMC approach, the Metropolis-Hastings algorithm, has been to estimate the posterior probability distribution over the space of total orderings, using the MML score (Korb and Nicholson 2011, Chap 8).

An approach to coping with search complexity is to use an anytime algorithm, that is, one which at any given time can be stopped to yield a best-so-far result. Yuan and Malone describe an anytime version of A^* search using metrics for BN discovery (Yuan and Malone 2013). The same authors show that a heuristic window A^* has reasonable efficiency; optimality is not guaranteed, but the algorithm can report a maximum distance from the optimal BN (Malone and Yuan 2013).

An alternative to model selection – searching for the single best model – is Bayesian model averaging, that is, searching for a set of models and weights for each of them Chickering and Heckerman (1997). And an alternative to *that* is to find a single Bayesian network that is equivalent to an averaged selection of networks (Dash and Cooper 2004).

Markov Blanket Discovery

Recently interest has grown in algorithms to learn, specifically, the Markov blankets around individual variables, which is a special kind of feature selection problem (see Feature Selection). This approach can help deal with "Big Data": whether the "curse of dimensionality" (too many variables) or extremely large data sets.

One use for this is in prediction: since the MB renders all other variables conditionally independent of a target variable, finding the MB means having all the variables required for an optimal predictor. Koller and Sahami developed an approximate Markov blanket filtering approach for prediction (Koller and Sahami 1996); Saeed improved the efficiency of this approach (Saeed 2008). Tsamardinos et al. describe the max-min hill climbing (MMHC) algorithm for MB discovery (Tsamardinos et al. 2006). Nägele et al. apply this to learning in very high-dimensional spaces (Nägele et al. 2007). Given the MB for a target variable, one can simply apply regression techniques (or any predictive technique) to the discovered variables. This works fine for standard prediction, but does not generalize to situations where some of the predictor variables are externally modified rather than observed. For an interesting collection of papers mostly applying some kind of Markov blanket discovery approach to prediction, see the collection (Guyon et al. 2008).

One can also apply MB discovery to causal learning, employing causal discovery within the reduced set of variables in the Markov blanket. Iterating this will yield multiple causal subnetworks, when a global causal network might be stitched together from them, as Aliferis et al. do with their HHC algorithm (Aliferis et al. 2010b), completing the whole causal discovery process while evading complexity problems. A current review of the issues and techniques can be found in two companion articles by Aliferis et al. (2010a, b).

Knowledge Engineering with Bayesian Networks

Another approach to dealing with the complexity and tribulations of global causal discovery is to aid the discovery process with prior information. Bayesian inference is, after all, done by combining priors with likelihoods, and the priors need not always be perfectly flavorless, such as uniform priors over the DAG space. In almost all applications where data threaten to overwhelm automated discovery, there is also at least some expertise, if only the ability to say, for example, that the sex of a patient is determined before adult lifestyle practices are adopted. Such temporal information provided to a discovery algorithm can provide a huge boost to the discovery process.

This quite simple kind of prior information, the temporal tiers within which the variables may be allocated, has been available in many of the discovery programs for a long time. PC, for example, allows tiers to be specified. K2 more restrictively required a total ordering of the variables. The methods described by Heckerman et al. (1994) go beyond tiers. They provide for the specification of a network or subnetwork; the prior probability of any network in the search space can be computed according to its distance from the network provided. They also introduced the idea of equivalent sample size, i.e., the weight to be given the prior information relative to the data, meaning that their priors are soft (probabilistic) rather than hard constraints. O'Donnell et al. (2006) adapted their MML score to allow soft priors for tiers, dependencies, direct and indirect causal relations, and networks or subnetworks, with variable degrees of confidence.

The flexible combination of prior information (expertise) with data in the causal discovery process allows for a fully fledged knowledge engineering process in the construction of Bayesian networks. Experts may be consulted for structural or parametric information, data may be gathered, and these different contributions may be weighted or reweighted according to the results of sensitivity analyses or other tests. The result can be a much faster and more useful approach to building and applying Bayesian networks. Research continues in this useful area of incorporating prior information, e.g., in Borboudakis et al. (2011).

Causal discovery with meaningful priors, by the way, shows that limiting discovery to patterns is insufficient: better priors, or better use of priors, can make a significant difference *within* patterns of DAGs.

Cross-References

- ▶ Anytime Algorithm
- ▶ Dimensionality Reduction
- ▶ Feature Selection
- ▶ Graphical Models
- ▶ Hidden Markov Models

Recommended Reading

The PC algorithm and variants were initially documented in Spirtes et al. (1993); their second edition Spirtes (2000) covers more ground. Their TETRAD V program is available from their web site http://www.phil.cmu.edu/projects/tetrad/. PC is contained within (and is available also with the weka machine learning platform at http://www.cs.waikato.ac.nz/ml/weka/).

A well-known tutorial by David Heckerman Heckerman (1999) (reprinted without change in Heckerman (2008)) is well worth looking at for background in causal discovery and parameterizing Bayesian networks. A more recent review of many of the topics introduced here is to be found in the article Daly et al. (2011). For other good treatments of parameterization, see Cowell et al. (1999) or Neapolitan (2003).

There are a number of useful anthologies in the area of learning graphical models. *Learning in Graphical Models* Jordan (1999) is one of the best, including Heckerman's tutorial and a variety of excellent reviews of causal discovery methods, such as Markov chain Monte Carlo search techniques.

Textbooks treating the learning of Bayesian networks include Borgelt and Kruse (2002); Neapolitan (2003); Korb and Nicholson (2011); Koller and Friedman (2009).

Aliferis CF, Statnikov A, Tsamardinos I, Mani S, Koutsoukos XD (2010) Local causal and Markov Blanket induction for causal discovery and feature selection for classification part I: algorithms and empirical evaluation. J Mach Learn Res 11:171–234

Aliferis CF, Statnikov A, Tsamardinos I, Mani S, Koutsoukos XD (2010) Local causal and Markov Blanket induction for causal discovery and feature selection for classification part II: analysis and extensions. J Mach Learn Res 11:235–284

Borboudakis G, Triantafillou S, Lagani V, Tsamardinos I (2011) A constraint-based approach to incorporate prior knowledge in causal models. In: European symposium on artificial neural networks, Computational Intelligence and machine learning, Bruges

Borgelt C, Kruse R (2002) Graphical models: methods for data analysis and mining. Wiley, New York

Bouckaert R (1993) Probabilistic network construction using the minimum description length principle. Lect Notes Comput Sci 747:41–48

Chickering DM (1995) A tranformational characterization of equivalent Bayesian network structures. In: Besnard P, Hanks S (eds) Proceedings of the 11th conference on uncertainty in artificial intelligence, San Francisco, pp 87–98

Chickering DM, Heckerman D (1997) Efficient approximations for the marginal likelihood of Bayesian networks with hidden variables. Mach Learn 29: 181–212

Chickering DM, Meek C (2002) Finding optimal Bayesian networks. In: Proceedings of the eighteenth annual conference on uncertainty in AI, San Francisco, pp 94–102

Chickering DM, Heckerman D, Meek C (2004) Large-sample learning of Bayesian networks is NP-hard. J Mach Learn Res 5:1287–1330

Chow C, Liu C (1968) Approximating discrete probability distributions with dependence trees. IEEE Trans Inf Theory 14:462–467

Cowell RG, Dawid AP, Lauritzen St L, Spiegelhalter DJ (1999) Probabilistic networks and expert systems. Springer, New York

Cruz-Ramírez N, Acosta-Mesa HG, Barrientos-Martínez RE, Nava-Fernández LA (2006) How good are the Bayesian information criterion and the Minimum Description length principle for model selection? A Bayesian network analysis. Lect Notes Comput Sci 4293:494–504

Daly R, Shen Q, Aitken S (2011) Learning Bayesian networks: approaches and issues. Knowl Eng Rev 26: 99–157

Dash D, Cooper GF (2004) Model averaging for prediction with discrete Bayesian networks. J Mach Learn Res 5:1177–1203

Donnell RO, Nicholson A, Han B, Korb K, Alam M, Hope L (2006) Causal discovery with prior information. In: Australasian joint conference on artificial intelligence, Auckland. Springer, pp 1162–1167

Guyon I, Aliferis C, Cooper G, Elisseeff A, Pellet J-P, Spirtes P, Statnikov A (eds) (2008) In: JMLR workshop and conference proceedings: causation and prediction challenge at WCCI 2008, Hong Kong, vol 3. Journal of Machine Learning Research

Heckerman D (1999) A tutorial on learning with Bayesian networks. In: Jordan M (ed) Learning in graphical models, pp 301–354. MIT Press, Cambridge

Heckerman D (2008) A tutorial on learning with Bayesian networks. In: Holmes DE, Jain LC (eds) Innovations in Bayesian networks. Springer, Berlin, pp 33–82

Heckerman D, Geiger D, Chickering DM (1994) Learning Bayesian networks: the combination of knowledge and statistical data. In: de Mantras L, Poole D (eds) Proceedings of the 10th conference on uncertainty in artificial intelligence, San Francisco, pp 293–301

Jordan MI (ed) (1999) Learning in graphical models. MIT Press, Cambridge

Koller D, Friedman N (2009) Probabilistic graphical models: principles and techniques. MIT, Cambridge

Koller D, Sahami M (1996) Toward optimal feature selection. In: Proceedings of the 13th international conference on machine learning, Bari, pp 284–292

Korb KB, Nicholson AE (2011) Bayesian artificial intelligence, 2nd edn. CRC Press, Boca Raton

Larrañaga P, Kuijpers CMH, Murga RH, Yurramendi Y (1996) Learning Bayesian network structures by searching for the best ordering with genetic algorithms. IEEE Trans Syst Man Cybern Part A 26: 487–493

Malone B, Yuan C (2013) Evaluating anytime algorithsm for learning optimal Bayesian networks. In: Nicholson A, Smyth P (eds) Proceedings of the 29th conference on uncertainty in AI, Bellevue, pp 381–390

Margaritis D, Thrun S (2000) Bayesian network induction via local neighborhoods. In: Solla SA, Leen TK, Müller KR (eds) Advances in neural information processing systems, vol 12. MIT Press, Cambridge, pp 505–511

Nägele A, Dejori M, Stetter M (2007) Bayesian substructure learning-approximate learning of very large network structures. In: Proceedings of the 18th European conference on machine learning, Warsaw. Lecture notes in AI, vol 4701, pp 238–249
Neapolitan RE (2003) Learning Bayesian networks. Prentice Hall, Harlow
Neil JR, Korb KB (1999) The evolution of causal models. In: Zhong N, Zhous L (eds) Third Pacific-Asia conference on knowledge discovery and datamining (PAKDD-99), Beijing. Springer, pp 432–437
Saeed M (2008) Bernoulli mixture models for markov blanket filtering and classification. In: Guyon I, Aliferis C, Cooper G, Elisseeff A, Pellet J-P, Spirtes P, Statnikov A (eds) JMLR workshop and conference proceedings: causation and prediction challenge (WCCI 2008), Hong Kong
Spirtes P, Glymour C, Scheines R (1993) Causation, prediction and search. Number 81 in lecture notes in statistics. Springer, New York
Spirtes P, Glymour C, Scheines R (2000) Causation, prediction and search, 2nd edn. MIT Press, Cambridge
Suzuki J (1996) Learning Bayesian belief networks based on the minimum description length principle. In: Saitta L (ed) Proceedings of the 13th international conference on machine learning, Bari. Morgan Kaufman, pp 462–470
Suzuki J (1999) Learning bayesian belief networks based on the MDL principle: an efficient algorithm using the branch and bound technique. IEEE Trans Inf Syst 82:356–367
Tsamardinos I, Brown LE, Aliferis CF (2006) The max-min hill-climbing Bayesian network structure learning algorithm. Mach Learn 65(1):31–78
Verma TS, Pearl J (1990) Equivalence and synthesis of causal models. In: Proceedings of the sixth conference on uncertainty in AI, Cambridge. Morgan Kaufmann, pp 220–227
Wallace CS, Korb KB, Dai H (1996) Causal discovery via MML. In: Saitta L (ed) Proceedings of the 13th international conference on machine learning, Bari. Morgan Kaufman, pp 516–524
Wong ML, Lam W, Leung KS (1999) Using evolutionary programming and minimum description length principle for data mining of Bayesian networks. IEEE Trans Pattern Anal Mach Intell 21(2): 174–178
Yuan C, Malone B (2013) Learning optimal Bayesian networks: a shortest path perspective. J Artif Intell Res 48:23–65

Learning in Logic

▶ Inductive Logic Programming

Learning in Worlds with Objects

▶ Relational Reinforcement Learning

Learning Models of Biological Sequences

William Stafford Noble[1] and Christina Leslie[2]
[1]Department of Genome Science/Department of Computer Science and Engineering, University of Washington, Seattle, WA, USA
[2]Memorial Sloan Kettering Cancer Research Center, New York, NY, USA

Abstract

The field of bioinformatics developed in the 1980s and 1990s, largely focusing on the computational analysis of newly available collections of DNA and protein sequences. In this context, a wide variety of machine learning analysis methods have been developed to understand the evolutionary history and molecular biology function of such sequences.

Definition

Hereditary information is stored in the nucleus of every living cell as biopolymers of deoxyribonucleic acids (DNA). DNA thus encodes the blueprint for all known forms of life. A DNA sequence can be expressed as a finite string over an alphabet of {A, C, G, T}, corresponding to the four DNA bases. The human genome consists of approximately three billion bases, divided among 23 chromosomes.

During its life, each cell makes temporary copies of short segments of DNA. These short-lived copies are comprised of ribonucleic acid (RNA). Each 3-mer of RNA can subsequently be translated, via the universal genetic code, into one of 20 amino acids. The resulting amino acid sequence is called a protein, and the DNA sequence that encodes the protein is called a gene.

Machine learning has been used to build models of many different types of biological sequences. These include models of short, functional elements within DNA or protein sequences, as well as models of genes, RNAs, and proteins.

Motivation and Background

Fundamentally, the motivation for building models of biological sequences is to understand the molecular mechanisms of the cell and the molecular basis for human disease. Each subheading below describes a different type of model, each of which attempts to capture a different facet of the underlying biology. All of these models, ultimately, aim to uncover either evolutionary or functional relationships among sequences.

Although DNA and protein sequences were available in small numbers as early as the 1950s, a significant number of sequences were not available until the 1980s. Most of the advances in model development started in the 1990s, with the exception of phylogenetic models, which were already being developed in the 1970s.

Structure of Learning System

Motifs

In the context of biological sequences, a "motif" is a short (typically 6–20 letters) subsequence that is functionally significant. A motif may correspond to, e.g., the location along the DNA strand where a particular protein binds or conversely the location along the protein that binds to the DNA. The motif can arise either via convergent evolution (when two sequences evolve to look similar to one another) or via evolutionary conservation (if sequences that lack the motif are likely to be eliminated via natural selection).

Motif discovery is the problem of identifying a previously unknown motif within a given collection of sequences, by finding patterns that occur more often than one would expect by chance. The problem is challenging in part because two occurrences of a given motif may not resemble each other exactly.

Work on motif discovery falls into two camps, based upon how the motifs themselves are represented. One camp uses position-specific scoring matrices (PSSMs), in which a motif of width w over an alphabet of size A is represented as a w-by-A probability matrix. In this matrix, each entry represents the probability that a given letter occurs at the given position. Early work in this area used expectation-maximization to identify protein motifs (Lawrence and Reilly 1990). This effort was significantly extended in the MEME algorithm (Bailey and Elkan 1994), which continues to be widely used today. A complementary approach uses Gibbs sampling (Lawrence et al. 1993), which offers several benefits, including avoiding local minima and the ability to sample multiple motifs simultaneously.

The other motif discovery camp uses a discrete motif representation, in which each motif is represented as a consensus sequence plus a specified maximum number of mismatches. In this formalism, enumerative methods can guarantee solving a given problem to optimality. For realistic problem sizes, this approach is most applicable to DNA, because of its much smaller alphabet size. One popular method of this kind is Weeder (Pavesi et al. 2004), which performed well in a comparison of motif discovery algorithms (Tompa et al. 2005).

More recently, the generation of large-scale in vitro binding data sets for DNA-binding proteins, for example, using protein binding microarray experiments, has renewed interest in algorithms for learning transcription factor binding motifs (Weirauch et al. 2013). These motifs may be represented either as PSSMs or by more general models based on k-length subsequences ("k-mers"). Meanwhile, experimental data on in vivo genomic binding of transcription factors based on chromatin immunoprecipitation followed by sequencing (ChIP-seq) has also been modeled using k-mer based learning methods (Arvey et al. 2012).

Proteins

A central problem in computational biology is the classification of proteins into functional and structural classes given their amino acid sequences. The 3D structure that a protein assumes after folding largely determines its function in the cell. However, directly obtaining a protein's 3D structure involves difficult experimental techniques such as X-ray crystallography or nuclear magnetic resonance, whereas it is relatively easy to determine a protein's sequence. Through evolution, structure is more conserved than sequence, so that detecting even very subtle sequence similarities, or remote homology, is important for predicting function.

Since the early 1980s, researchers have developed a battery of successively more powerful methods for detecting protein sequence similarities. This development can be broken into three main stages. Early methods focused on the *pairwise comparison* problem and assessed the statistical significance of similarities between two proteins based on pairwise alignment. These methods are only capable of recognizing relatively close homologies. The BLAST algorithm (Altschul et al. 1990), based on heuristic alignment, and related tools are the most widely used methods for pairwise sequence comparison and database search today.

In the second stage, further accuracy was achieved by collecting aggregate statistics from a set of similar sequences and comparing the resulting statistics to a single, unlabeled protein of interest. One important example of *family-based models* are profile hidden Markov models (HMMs) (Krogh et al. 1994), probabilistic generative models estimated from a multiple alignment of sequences from a protein family. Profile HMMs generate variable length sequences by allowing insertions and deletions relative to the core residues of the alignment.

The third stage introduced *discriminative algorithms* based on classifiers like support vector machines for protein classification and remote homology detection. Such methods train both on positive sequences belonging to a protein family as well as negative examples consisting of sequences unrelated to the family. They require protein sequences to be represented using an explicit feature mapping or a kernel function in order to train the classifier. The first discriminative protein classification algorithm was the SVM-Fisher method (Jaakkola et al. 2000), which uses a profile HMM to extract a feature vector of Fisher scores for each input sequence x, defined by the gradient vector

$$\nabla_\theta \log P(x|\theta)|_{\theta=\theta_0},$$

where $\log P(x|\theta)$ is the log likelihood function the sequence relative to the HMM and θ_0 is the maximum likelihood estimate for the model parameters. Another feature representation that has been used is the empirical kernel map:

$$\Phi(x) = \langle s(x_1, x), \ldots, s(x_m, x) \rangle,$$

where $s(x, y)$ is a function depending on a pairwise similarity score between x and y and x_i, $i = 1 \ldots m$, are the training sequences (Liao and Noble 2002). In addition, it is possible to construct useful kernels directly without explicitly depending on generative models by using subsequence-based string kernels. For example, the mismatch kernel (Leslie et al. 2003) is defined by a histogram-like feature map The feature space is indexed by all possible k-length subsequences $\alpha = a_1 a_2 \ldots a_k$, where each a_i is a character in the alphabet $\mathcal{A}$ of amino acids. The feature map is defined on k-gram α by $\Phi(\alpha) = (\phi_\beta(\alpha))_{\mathcal{A}^k}$ where $\phi_\beta(\alpha) = 1$ if α is within m mismatches of β, 0 otherwise, and is extended additively to longer sequences: $\Phi(x) = \sum_{k-\text{grams} \in x} \Phi(\alpha)$.

One challenge in using the k-gram representations is that computing sequence similarity using this feature map can quickly become computationally intractable (Leslie et al. 2003), particularly when the number of mismatches m is large or if the sequence alphabet grows beyond the 20 basic amino acids to encode, for instance, additional physical or chemical properties of the protein sequence. To handle this problem, Kuksa et al. (2008) introduced linear time algorithms with alphabet-independent complexity applicable to computation of a large class of existing string kernels. The approach relies on the ability to precompute, in closed form, the number of k-grams that are at most m mismatches away from two short strings α and β. These methods have been subsequently extended from Hamming (i.e., match or no match) to arbitrary measures of similarity $\mathcal{S}(a, b)$ between elements a,b of each k-gram (Kuksa et al. 2012).

Genes

After a genome (or a portion of a genome) has been sequenced, a biologist's first question is usually, "Where are the genes?" In simple organisms, most of the genome is translated into proteins, and so the gene-finding problem reduces, essentially, to identifying the boundaries between genes. In more complex organisms, a large proportion of the genome is comprised of nonprotein coding DNA. The human genome, for example, is comprised of approximately 98 % noncoding DNA. This noncoding DNA is interspersed between coding regions and even in the midst of a single coding region. The gene-finding problem, canonically, is to identify the regions of a given DNA sequence that encode proteins.

Initial methods for gene finding combined scores produced by different types of detectors. A *signal* detector attempts to recognize local, fixed-length features, such as characterize the boundaries between coding and noncoding regions within a single gene. A *content* detector attempted to recognize larger patterns on the basis of compositional statistics. Early gene-finding algorithms combined these various scores in an ad hoc fashion to identify gene-like regions.

In the mid-1990s, several research groups began using HMMs for gene finding. HMMs provide a coherent, fully probabilistic method that is capable of capturing many of the complexities of real genes. An early, widely used method was Genscan (Burge and Karlin 1997), which uses fifth-order Markov statistics along with variable duration HMMs. Next-generation gene finders used conditional random field models (Bernal et al. 2007) and large-margin structured output techniques (Rätsch et al. 2007).

A more recent, unsupervised variant of the gene-finding problem is *semiautomated genome annotation* (Day et al. 2007). In this case, the input is not the DNA sequence per se but a collection of sequence-based measurements arrayed along the genome, representing local DNA conformation as well as properties of proteins bound to the DNA. The task is to simultaneously partition the genome and assign an integer label to each segment in such a way that segments with the same label have similar data. The process is semiautomated because the semantics of the labels – corresponding to genes, regulatory elements, etc. – must be inferred manually in a post-processing step.

RNAs

Most RNA molecules are so-called messenger RNAs, which are used in the production of a corresponding protein molecule. Some RNAs, however, do not code for proteins but instead function on their own. These RNAs fall into functional categories, but they are not easily recognized by HMMs because (1) the RNAs themselves are often very short, and (2) functional RNA typically folds up in a deterministic fashion and therefore exhibits nonlocal dependencies along the RNA sequence.

Useful RNA modeling is therefore accomplished using covariance models, which are a subclass of stochastic context-free grammars. The foundational work in this area was due to Eddy and Durbin (1994), who addressed both the structure inference problem and the inference of transition and emission probabilities given the structure. They applied these algorithms to transfer RNAs (tRNAs), and the approach was the basis for widely used tools such as Rfam.

Much effort in RNA covariance models has been devoted to improving the time and space efficiency of the algorithms associated with covariance models. For example, Eddy (2002) introduced a memory-efficient variant of the core dynamic programming algorithm used to align a covariance model to an RNA sequence. This improvement was practically important, since it reduced the $O(N^3)$ space requirement for a length N RNA sequence. Other work has focused on accelerating database search using the modeled families.

Recent efforts have focused on algorithms for genome-wide screens to discover functional noncoding RNAs as well as small regulatory

RNAs like microRNAs. Various approaches to this problem have incorporated conservation as well as RNA structure prediction, both using covariance models and other methodologies. One such algorithm is RNAz (Washietl et al. 2005), which combines a measure for thermodynamic stability with a measure for structure conservation in an SVM approach to detect functional RNAs in multiple sequence alignments.

Phylogenetic Models

Phylogenetic models attempt to infer the series of evolutionary events (mutations, insertions, deletions, etc.) that gave rise to an observed collection of DNA or protein sequences. In most cases, these models ignore the possibility of copying DNA between individuals or species, and therefore represent the history as a phylogenetic tree, in which leaf nodes represent the observed sequences and the internal nodes represent unobserved ancestral sequences. Of primary interest is inferring the topology and branch lengths of this tree.

Methods for phylogenetic tree inference can be divided into three classes: parsimony, distance, and likelihood methods, all described in detail in Felsenstein (2003).

Parsimony methods search for a tree that requires the smallest number of mutations, insertions, or deletions along its branches. Because the search space of possible tree topologies is so large, this approach is feasible only for relative small sets of sequences – tens rather than hundreds. Also, because parsimony models do not allow for so-called back mutations – where a letter mutates to a different letter and then back again – and other similar events, parsimony models are provably suboptimal for distantly related sequences.

Distance methods replace parsimony with a generalized notion of distance, which may include back mutation. A series of increasingly sophisticated distance metrics have been developed in this domain, starting with the one-parameter Jukes-Cantor model and the two-parameter Kimura model. Given an all-versus-all distance matrix, various tree inference algorithms can be used, including neighbor joining and agglomerative hierarchical clustering (called UPGMA in phylogenetics).

The third class of models use a fully probabilistic approach and attempt to infer the tree with maximum likelihood, given the observed sequences. This approach was first outlined in 1973 (Felsenstein 1973), but was not computationally feasible for large sets of sequences until recently. Current methods employ Markov chain Monte Carlo methods to carry out the search.

More recently, the so-called alignment-free methods (Kuksa and Pavlovic 2009) have been considered for the narrower problem of species identification in the context of DNA barcoding. DNA barcoding was introduced as a taxonomic tool for characterizing species using fragments of a DNA sequence from standard gene regions, such as the mitochondrial DNA (mtDNA) (Hebert et al. 2003). These alignment-free methods are similar in spirit to the discriminative kernel approaches used for protein classification. They avoid the costly process of explicit phylogenetic tree-building and instead focus on more scalable identification of few species families.

Programs and Data

Following are some of the more popular web sites for performing biological sequence analysis:

- BLAST and PSI-BLAST (http://www.ncbi.nlm.nih.gov/BLAST) search a protein or DNA sequence with a given, query sequence, and return a ranked list of homologs.
- MEME (http://meme.sdsc.edu) searches a given set of DNA or protein sequences for one or more recurrent motif patterns.
- HMMER (http://hmmer.janelia.org) is an HMM toolkit for training and searching with profile HMMs of proteins.
- Pfam (http://pfam.janelia.org) is a searchable library of profile HMMs corresponding to a curated collection of homologous protein domains.
- Rfam (http://rfam.janelia.org) is an analogous database of multiple sequence alignments and

covariance models covering many common noncoding RNA families.

- PHYLIP (http://evolution.genetics.washington.edu/phylip.html) is a free software toolkit that includes many common phylogenetic inference algorithms.

Recommended Reading

Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ (1990) A basic local alignment search tool. J Mol Biol 215:403–410

Arvey A, Agius P, Noble WS, Leslie C (2012) Sequence and chromatin determinants of cell-type specific transcription factor binding. Genome Res 22(9):1723–1734. PMC3431489

Bailey TL, Elkan CP (1994) Fitting a mixture model by expectation-maximization to discover motifs in biopolymers. In: Altman R, Brutlag D, Karp P, Lathrop R, Searls D (eds) Proceedings of the second international conference on intelligent systems for molecular biology. AAAI Press, pp 28–36

Bernal A, Crammer K, Hatzigeorgiou A, Pereira F (2007) Global discriminative learning for higher-accuracy computational gene prediction. PLoS Comput Biol 3(3):e54

Burge C, Karlin S (1997) Prediction of complete gene structures in human genomic DNA. J Mol Biol 268(1):78–94

Day N, Hemmaplardh A, Thurman RE, Stamatoyannopoulos JA, Noble WS (2007) Unsupervised segmentation of continuous genomic data. Bioinformatics 23(11):1424–1426

Eddy SR (2002) A memory-efficient dynamic programming algorithm for optimal alignment of a sequence to an rna secondary structure. BMC Bioinfo 3:18

Eddy SR, Durbin R (1994) RNA sequence analysis using covariance models. Nucleic Acids Res 22:2079–2088

Felsenstein J (1973) Maximum-likelihood estimation of evolutionary trees from continuous characters. Am J Hum Genet 25:471–492

Felsenstein J (2003) Inferring phylogenies. Sinauer Associates, Sunderland

Hebert PDN, Cywinska A, Ball SL, deWaard JR (2003) Biological identifications through DNA barcodes. Proc Biol Sci/R Soc 270(1512):313–321

Jaakkola T, Diekhans M, Haussler D (2000) A discriminative framework for detecting remote protein homologies. J Comput Biol 7(1–2):95–114

Krogh A, Brown M, Mian I, Sjolander K, Haussler D (1994) Hidden Markov models in computational biology: applications to protein modeling. J Mol Biol 235:1501–1531

Kuksa P, Pavlovic V (2009) Efficient alignment-free DNA barcode analytics. BMC Bioinform 10(Suppl 14):S9

Kuksa P, Huang P-H, Pavlovic V (2008) Scalable algorithms for string Kernels with inexact matching. In: Proceedings neural information processing systems, Vancouver, Dec 2008

Kuksa P, Khan I, Pavlovic V (2012) Generalized similarity kernels for efficient sequence classification. In: SIAM international conference on data mining. SIAM, pp 873–882

Lawrence CE, Reilly AA (1990) An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. Proteins 7(1): 41–51

Lawrence CE, Altschul SF, Boguski MS, Liu JS, Neuwald AF, Wootton JC (1993) Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. Science 262(5131):208–214. Web server at http://bayesweb.wadsworth.org/gibbs/gibbs.html

Leslie C, Eskin E, Weston J, Noble WS (2003) Mismatch string kernels for SVM protein classification. In: Becker S, Thrun S, Obermayer K (eds) Advances in neural information processing systems, Cambridge. MIT, pp 1441–1448

Liao L, Noble WS (2002) Combining pairwise sequence similarity and support vector machines for remote protein homology detection. In: Proceedings of the sixth annual international conference on computational molecular biology, Washington, DC, Apr 18–21, pp 225–232

Pavesi G, Mereghetti P, Mauri G, Pesole G (2004) Weeder Web: discovery of transcription factor binding sites in a set of sequences from co-regulated genes. Nucleic Acids Res 32(Web server issue):W199–203. Web server at http://159.149.160.51/modtools/

Rätsch G, Sonnenburg S, Srinivasan J, Witte H, Müller KR, Sommer R, Schölkopf B (2007) Improving the *c. elegans* genome annotation using machine learning. PLoS Comput Biol 3(2):e20

Tompa M, Li N, Bailey TL, Church GM, De Moor B, Eskin E, Favorov AV, Frith MC, Fu Y, Kent WJ, Makeev VJ, Mironov AA, Noble WS, Pavesi G, Pesole G, Regnier M, Simonis N, Sinha S, Thijs G, van Helden J, Vandenbogaert M, Weng Z, Workman C, Ye C, Zhu Z (2005) Assessing computational tools for the discovery of transcription factor binding sites. Nat Biotechnol 23(1):137–144

Washietl S, Hofacker IL, Stadler PF (2005) Fast and reliable prediction of noncoding RNAs. Proc Natl Acad Sci USA 102(7):2454–2459

Weirauch MT, Cote A, Norel R, Annala M, Zhao Y, Riley TR, Saez-Rodriguez J, Cokelaer T, Vedenko A, Talukder S, DREAM5 Consortium (including W. S. Noble), Bussemaker HJ, Morris QD, Bulyk ML, Stolovitzky G, Hughes TR (2013) Evaluation of methods for modeling transcription factor sequence specificity. Nat Biotechnol 31(2):126–134. PMC3687085

Learning to Learn

► Metalearning

Learning to Rank

Hang Li
Huawei Technologies, Hong Kong, China

Abstract

Many tasks in information retrieval, natural language processing, and data mining are essentially ranking problems. These include document retrieval, expert search, question answering, collaborative filtering, and keyphrase extraction. Learning to rank is a subarea of machine learning, studying methodologies and theories for automatically constructing a model from data for a ranking problem (Liu T-Y, Found Trends Inf Retr 3(3):225–331, 2009; Li H, Synth Lect Hum Lang Technol 4(1):1–113, 2011a; Li H, IEICE Trans Inf Syst 94-D(10):1854–1862, 2011b).

Learning to rank is usually formalized as a supervised learning task, while unsupervised learning and semi-supervised learning formulations are also possible. In learning, training data consisting of sets of objects as well as the total or partial orders of the objects in each set is given, and a ranking model is learned using the data. In prediction, a new set of objects is given, and a ranking list of the objects is created using the ranking model.

Learning to rank has been intensively studied in the past decade and many methods of learning to rank have been proposed. Popular methods include Ranking SVM, IR SVM, AdaRank, LambdaRank, and LambdaMART. The methods can be categorized into the pointwise, pairwise, and listwise approaches according to the loss functions which they use. It is known that learning-to-rank methods, such as LambdaMART, are being employed in a number of commercial web search engines.

In this entry, we describe the formulation as well as several methods of learning to rank. Without loss of generality, we take document retrieval as example.

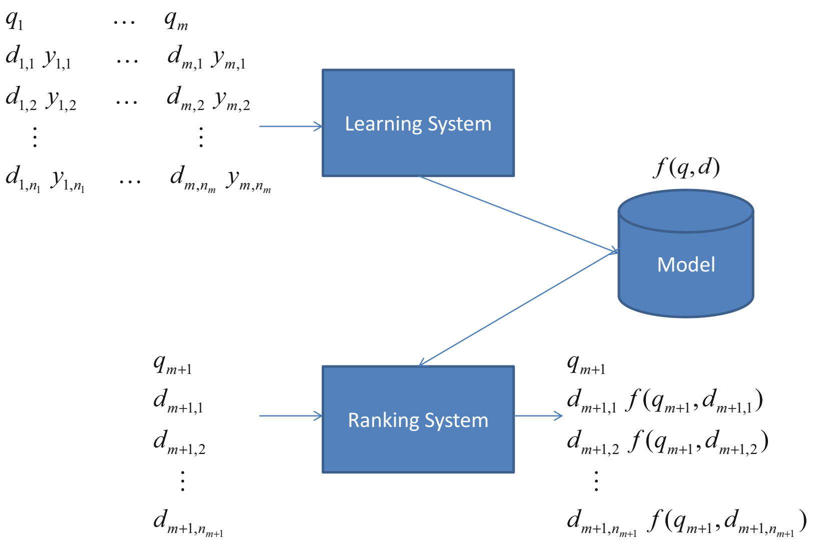
Solution

Problem Formulation

In the supervised learning setting, learning to rank includes training and testing phases (see Fig. 1). In training, the learning system learns a ranking model from given training data, and in testing, given a query the ranking system assigns scores to documents with respect to the query using the ranking model and sorts the documents on the basis of the scores.

The training data consists of queries and documents. Each query is associated with a number of documents. The relevance of the documents with respect to the query is also given. The relevance can be represented in several ways. Here, we take the most widely used approach and assume that the relevance of a document with respect to a query is represented by a label, while the labels denote several grades (levels). The higher grade a document has, the more relevant the document is.

Suppose that $\mathcal{Q}$ is the query set and $\mathcal{D}$ is the document set. Suppose that $\mathcal{Y} = \{1, 2, \cdots, l\}$ is the label set, where labels represent grades. There exists a total order between the grades $l \succ l-1 \succ \cdots \succ 1$, where $\succ$ denotes the order relation. Further suppose that $Q = \{q_1, q_2, \cdots, q_m\}$ is the set of queries for training and q_i is the i-th query. $D_i = \{d_{i,1}, d_{i,2}, \cdots, d_{i,n_i}\}$ is the set of documents associated with query q_i, and $\mathbf{y}_i = \{y_{i,1}, y_{i,2}, \cdots, y_{i,n_i}\}$ is the set of labels associated with query q_i, where n_i denotes the sizes of D_i and $\mathbf{y}_i$; $d_{i,j}$ denotes the j-th document in D_i; and $y_{i,j} \in Y$ denotes the j-th grade


q_1 $\cdots$ q_m
$d_{1,1}$ $y_{1,1}$ $\cdots$ $d_{m,1}$ $y_{m,1}$
$d_{1,2}$ $y_{1,2}$ $\cdots$ $d_{m,2}$ $y_{m,2}$
d_{1,n_1} y_{1,n_1} $\cdots$ d_{m,n_m} y_{m,n_m}
Learning System
$f(q,d)$
Model
q_{m+1}
$d_{m+1,1}$
$d_{m+1,2}$
$d_{m+1,n_{m+1}}$
Ranking System
q_{m+1}
$d_{m+1,1}$ $f(q_{m+1}, d_{m+1,1})$
$d_{m+1,2}$ $f(q_{m+1}, d_{m+1,2})$
$d_{m+1,n_{m+1}}$ $f(q_{m+1}, d_{m+1,n_{m+1}})$

Learning to Rank, Fig. 1 An overview of learning to rank for document retrieval

label in $\mathbf{y}_i$. The original training set is denoted as $S = \{(q_i, D_i), \mathbf{y}_i\}_{i=1}^{m}$.

A feature vector $x_{i,j} = \phi(q_i, d_{i,j})$ is created from each query-document pair $(q_i, d_{i,j})$, $i = 1, 2, \cdots, m; j = 1, 2, \cdots, n_i$, where ϕ denotes the feature functions. That is to say, features are defined as functions of a query-document pair. For example, BM25 and PageRank are typical features. Letting $\mathbf{x}_i = \{x_{i,1}, x_{i,2}, \cdots, x_{i,n_i}\}$, the training data set is also represented as $S' = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{m}$. Here $x \in \mathcal{X}$ and $\mathcal{X} \subseteq \Re^d$, where d denotes the number of features.

We aim to train a (local) ranking model $f(q,d) = f(x)$ that can assign a score to a given query-document pair q and d or equivalently to a given feature vector x. More generally, we can also consider training a global ranking model $F(q, D) = F(\mathbf{x})$.

Let the documents in D_i be identified by the integers $\{1, 2, \cdots, n_i\}$. We define a permutation (ranking list) $\boldsymbol{\pi}_i$ on D_i as a bijection from $\{1, 2, \cdots, n_i\}$ to itself. We use Π_i to denote the set of all possible permutations on D_i and use $\pi_i(j)$ to denote the rank (or position) of the j-th document (i.e., $d_{i,j}$) in permutation π_i. Ranking is nothing but to select a permutation $\pi_i \in \Pi_i$ for the given query q_i and the associated documents D_i using the scores given by the ranking model $f(q_i, d_i)$.

The test data consists of a new query q_{m+1} and associated documents D_{m+1}. We create feature vector $\mathbf{x}_{m+1}$, use the trained ranking model to assign scores to the documents D_{m+1}, sort them based on the scores, and give the ranking list of documents as output π_{m+1}.

The training and testing data is similar to, but different from, the data in conventional supervised learning such as classification and regression. Query and its associated documents form a group. The groups are i.i.d. data, while the instances within a group are not i.i.d. data. A local ranking model is a function of a query and a document or, equivalently, a function of a feature vector derived from a query and a document.

Evaluation on the performance of a ranking model is carried out by comparison between the ranking lists output by the model and the ranking lists given as the ground truth. Several evaluation measures are usually used in information retrieval, such as NDCG (Normalized Discounted Cumulative Gain), DCG (Discounted Cumulative Gain), MAP (Mean Average Precision), and Kendall's Tau.

Evaluation Measures: DCG and NDCG

We give definitions of DCG and NDCG; both are most utilized in document retrieval. NDCG is normalized and thus it is easier to use NDCG in comparison.

Given query q_i and associated documents D_i, suppose that π_i is the ranking list (permutation) on D_i and $\mathbf{y}_i$ is the set of labels (grades) of D_i. DCG measures the goodness of the ranking list with the labels. Specifically, DCG at position k is defined as

$$DCG(k) = \sum_{j:\pi_i(j)\leq k} G(j)D(\pi_i(j)),$$

where $G_i(\cdot)$ is a gain function, $D_i(\cdot)$ is a position discount function, $\pi_i(j)$ is the position of $d_{i,j}$ in π_i, and the summation is taken over the top k positions in the ranking list π_i. The gain function is normally defined as an exponential function of grade

$$G(j) = 2^{y_{i,j}} - 1,$$

where $y_{i,j}$ is the label (grade) of document $d_{i,j}$ in ranking list π_i. The position discount function is normally defined as a logarithmic function of position

$$D(\pi_i(j)) = \frac{1}{\log_2(1+\pi_i(j))},$$

where $\pi_i(j)$ is the position of document $d_{i,j}$ in ranking list π_i. DCG represents the cumulative gain of accessing the information from position one to position k with discounts on the positions. Hence, DCG at position k becomes

$$\mathrm{DCG}(k) = \sum_{j:\pi_i(j)\leq k} \frac{2^{y_{i,j}} - 1}{\log_2(1+\pi_i(j))}.$$

NDCG is normalized DCG, and NDCG at position k is defined as

$$\mathrm{NDCG}(k) = G^{-1}_{\max,i}(k)DCG(k),$$

where $G_{\max,i}(k)$ is the normalizing factor and is chosen such that a perfect ranking π_i^*'s NDCG score at position k is 1. In a perfect ranking, the documents with higher grades are always ranked higher. Note that there can be multiple perfect rankings for a query and associated documents. Then, NDCG at position k becomes

$$\mathrm{NDCG}(k) = G^{-1}_{\max,i}(k) \sum_{j:\pi_i(j)\leq k} \frac{2^{y_{i,j}} -1}{\log_2(1+\pi_i(j))}.$$

In evaluation, DCG and NDCG values are further averaged over queries.

Objective Function of Learning

Learning to rank is generally formalized as a supervised learning task. Suppose that $\mathcal{X}$ is the input space (feature space) consisting of lists of feature vectors and $\mathcal{Y}$ is the output space consisting of lists of grades. Further suppose that $\mathbf{x}$ is an element of $\mathcal{X}$ representing a list of feature vectors and $\mathbf{y}$ is an element of $\mathcal{Y}$ representing a list of grades. Let $P(X,Y)$ be an unknown joint probability distribution where random variable X takes $\mathbf{x}$ as its value and random variable Y takes $\mathbf{y}$ as its value.

Assume that $F(\cdot)$ is a function mapping from a list of feature vectors $\mathbf{x}$ to a list of scores. The goal of the learning task is to automatically learn a function $\hat{F}(\mathbf{x})$ given training data $(\mathbf{x}_1,\mathbf{y}_1),(\mathbf{x}_2,\mathbf{y}_2),\ldots,(\mathbf{x}_m,\mathbf{y}_m)$. Each training instance is comprised of feature vectors $\mathbf{x}_i$ and the corresponding grades $\mathbf{y}_i$ $(i = 1,\cdots,m)$. Here m denotes the number of training instances.

$F(\mathbf{x})$ and $\mathbf{y}$ can be further written as $F(\mathbf{x}) = (f(x_1), f(x_2),\cdots,f(x_n))$ and $\mathbf{y} = (y_1,y_2,\cdots,y_n)$. The feature vectors represent objects to be ranked. Here $F(\mathbf{x})$ denotes the global ranking function, $f(x)$ denotes the local ranking function, and n denotes the number of feature vectors and grades.

A loss function $L(\cdot,\cdot)$ is utilized to evaluate the prediction result of $F(\cdot)$. First, feature vectors $\mathbf{x}$ are ranked according to $F(\mathbf{x})$, and then the top n results of the ranking are evaluated using their corresponding grades $\mathbf{y}$. If the feature vectors with higher grades are ranked higher, then the loss will be small. Otherwise, the loss will be large. The loss function is specifically represented as $L(F(\mathbf{x}), \mathbf{y})$. Note that the loss function for ranking is slightly different from the loss functions in other statistical learning tasks, in the sense that it makes use of sorting.

The loss function can be defined, for example, based on NDCG (Normalized Discounted Cumulative Gain) and MAP (Mean Average Precision). In the former case, we have

$$L(F(\mathbf{x}), \mathbf{y}) = 1 - \text{NDCG}.$$

The risk function, i.e., the objective function in learning, is further defined as the expected loss function with respect to the joint distribution $P(X, Y)$:

$$R(F) = \int_{\mathcal{X} \times \mathcal{Y}} L(F(\mathbf{x}), \mathbf{y}) dP(\mathbf{x}, \mathbf{y}).$$

Given training data, the empirical risk function is defined as follows:

$$\hat{R}(F) = \frac{1}{m} \sum_{i=1}^{m} L(F(\mathbf{x}_i), \mathbf{y}_i). \tag{1}$$

The learning task then becomes minimization of the empirical risk function, as in other learning tasks. The minimization could be difficult due to the nature of the loss function (it is not continuous and it uses sorting). We can consider using a surrogate loss function $L'(F(\mathbf{x}), \mathbf{y})$.

The corresponding empirical risk function is defined as follows:

$$\hat{R}'(F) = \frac{1}{m} \sum_{i=1}^{m} L'(F(\mathbf{x}_i), \mathbf{y}_i). \tag{2}$$

One can also introduce a regularizer to the minimization. In such case, the learning problem becomes minimization of the regularized empirical risk function based on the surrogate loss.

For the surrogate loss function, there are also different ways to define it, which lead to different approaches to learning to rank, referred to as pointwise loss, pairwise loss, and listwise loss functions.

Methods

Ranking SVM and IR SVM

A perfect ranking implies perfect classification on all pairs of objects, and an imperfect ranking implies existence of incorrect classification on pairs of objects. Therefore, learning of a ranking model can be transformed to learning of a pairwise classification or regression model. This is the idea behind the pairwise approach to learning to rank. Ranking SVM proposed by Herbrich et al. (1999) is a typical example.

Ranking SVM takes pairs of objects (feature vectors) at different grades as new objects and takes the orders of the pairs of objects as positive or negative classes. It learns an SVM classifier which can classify the order of pair of objects and then utilizes the SVM classifier to assign scores to newly given objects and rank the objects on the basis of the scores. One can easily verify that if the classifier is a linear function, then it can be directly utilized as a ranking model.

It can be proved that the "pairwise" loss function in Ranking SVM is an upper bound of (1-NDCG) Li (2011a), and therefore Ranking SVM is a method equivalent to minimizing empirical risk based on a surrogate loss function (cf., Eq. (2)).

IR SVM proposed by Cao et al. (2006) is an extension of Ranking SVM for information retrieval (IR). Ranking SVM transforms ranking into pairwise classification, and thus it actually makes use of the 0-1 loss in the classification. There exists a gap between the loss function and the IR evaluation measures such as NDCG. IR SVM attempts to bridge the gap by modifying the 0-1 loss, that is, conducting cost-sensitive learning of Ranking SVM.

One problem with Ranking SVM is that it equally treats document pairs across different grades (levels). Another issue with Ranking SVM is that it equally treats document pairs from different queries. IR SVM addresses the above two problems by modifying the hinge loss function.

Specifically, it sets different losses for document pairs across different grades and from different queries. To emphasize the importance of correct ranking on the top, the loss function heavily penalizes errors related to the top. To increase the influence of queries with less documents, the loss function heavily penalizes errors from the queries.

AdaRank

Since the evaluation measures in IR are defined on lists of objects, it is more natural and effective to directly optimize "listwise" loss functions defined on lists. This is the presumption in the listwise approach to learning to rank. AdaRank, proposed by Xu and Li (2007), is one of the listwise algorithms. One advantage of AdaRank is its simplicity, and it is perhaps one of the simplest learning-to-rank algorithms.

In learning, ideally we would create a ranking model that can maximize the ranking accuracy in terms of an evaluation measure (e.g., NDCG) on the training data or equivalently minimize the empirical risk function in Eq. (1) specified as

$$\sum_{i=1}^{m}(1 - E(\pi_i, \mathbf{y}_i)),$$

where $\mathbf{x}_i$ is a list of feature vectors, $\mathbf{y}_i$ is the corresponding list of grades, π_i is the permutation of feature vectors $\mathbf{x}_i$ by the ranking model $f(x)$, $E(\pi_i, \mathbf{y}_i)$ is the evaluation result of π_i based on $\mathbf{y}_i$ in terms of the evaluation measure, and m is the number of training instances.

The empirical risk function is not smooth and differentiable, and thus straightforward optimization of the evaluation result might not work. Instead, we can consider optimizing an upper bound of the function. For example, one upper bound is the following function:

$$\sum_{i=1}^{m}\exp(-E(\pi_i, \mathbf{y}_i)),$$

which is continuous, differentiable, and even convex with respect to E.

AdaRank minimizes the upper bound, by taking the boosting approach. Mimicking the famous AdaBoost algorithm, AdaRank conducts stepwise minimization of the upper bound. More specifically, AdaRank repeats the process of reweighting the training instances, creating a weak ranker, and assigning a weight to the weak ranker. Finally, AdaRank linearly combines the weak rankers as the ranking model.

One can prove that AdaRank can continuously reduce the empirical risk during the training process, under certain conditions. When the evaluation measure is dot product, AdaRank can reduce the risk to zero.

LambdaRank and LambdaMART

The objective function (empirical risk function) in learning to rank is not continuous and differentiable, and it depends on sorting. This makes it difficult to use gradient decent to optimize the function. LambdaRank and LambdaMART, proposed by Burges (2010), manage to solve the problem by directly defining and utilizing a gradient function of the risk function.

Suppose that the ranking model, query, and documents are given. Then each document receives a score from the ranking model, and a ranking list can be created by sorting the documents based on the scores. Since the documents are also assigned ground truth labels, a ranking evaluation result based on an IR measure can be obtained. Suppose that we use a surrogate loss function L to approximate the IR evaluation measure. Then, an evaluation result based on the surrogate loss function L can also be obtained. It is this evaluation result which LambdaRank attempts to continuously optimize.

LambdaRank does not explicitly give the definition of the loss function. Instead it defines the *gradient* function of the surrogate loss function. More specifically, the gradient function is defined as

$$\frac{\partial L}{\partial s_i} = -\lambda_i(s_1, y_1, \cdots, s_n, y_n),$$

where $s_1, s_2, \cdots, s_n$ denote the scores of documents and $y_1, y_2, \cdots, y_n$ denote the labels of documents. Note that the index i is on a single document. That is to say, the gradient of a document depends on the scores and labels of the other documents. The sign is chosen such that a positive value for a document means that the document must reduce the loss. The gradients of documents are calculated after the current model generates a ranking list of documents for the query. The negative gradient function is called lambda function, and that is why the method is called LambdaRank. LambdaRank utilizes a neural network as its ranking model.

LambdaMART follows the idea of LambdaRank, but it utilizes an ensemble of trees as its ranking model and employs the Gradient Tree Boosting algorithm to build the ranking model. Specifically, LambdaMART directly uses the lambda function as the gradient function in the learning process of Gradient Tree Boosting.

In the Yahoo Learning to Rank Challenge, LambdaMART achieved the best performance. It is viewed as one of the state-of-the-art methods for learning to rank and is being used in a number of commercial search systems.

Applications

Learning to rank has been successfully applied to a wide variety of applications, including document retrieval, expert search, definition search, personalized search, online advertisement, collaborative filtering, question answering, keyphrase extraction, document summarization, and machine translation. Particularly, in document retrieval there are many signals which can represent relevance. Incorporating such information into the ranking model and automatically constructing the ranking model by using data become a natural choice. In fact, learning to rank has become one of the fundamental technologies for document retrieval.

Recommended Reading

Burges CJC (2010) From RankNet to LambdaRank to LambdaMART: an overview. Microsoft Research Technical Report, MSR-TR-2010-82

Cao Y, Xu J, Liu T-Y, Li H, Huang Y, Hon H-W (2006) Adapting ranking SVM to document retrieval. In: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval, Seattle, pp 186–193

Herbrich R, Graepel T, Obermayer K (1999) Large margin rank boundaries for ordinal regression. Adv Neural Inf Process Syst 115–132

Li H (2011a) Learning to rank for information retrieval and natural language processing. Synth Lect Hum Lang Technol 4(1):1–113

Li H (2011b) A short introduction to learning to rank. IEICE Trans Inf Syst 94-D(10):1854–1862

Liu T-Y (2009) Learning to rank for information retrieval. Found Trends Inf Retr 3(3):225–331

Xu J, Li H (2007) Adarank: a boosting algorithm for information retrieval. In: Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval, Amsterdam, pp 391–398

Learning Using Privileged Information

Viktoriia Sharmanska[1] and Novi Quadrianto[2]
[1]Department of Informatics, University of Sussex, SMiLe CLiNiC, Falmer, UK
[2]Department of Informatics, SMiLe CLiNiC, University of Sussex, Brighton, UK

Abstract

When applying machine learning techniques to real-world problems, prior knowledge plays a crucial role in enriching the learning system. This prior knowledge is typically defined by domain experts and can be integrated into machine learning algorithms in a variety of ways: as a preference of certain prediction functions over others, as a Bayesian prior over parameters, or as additional information about the samples in the training set used for learning a prediction function. The latter setup is called *learning using privileged information (LUPI)* and was adopted by Vapnik and Vashist in (Neural Netw, 2009). Formally, LUPI refers to the setting when, in addition to the main data modality, the learning system has access to an extra source of information

about the training examples. The additional source of information is only available during training and therefore is called privileged. The main goal of LUPI is to utilize privileged information and to learn a better model in the main data modality than one would learn without the privileged source. As an illustration, for protein classification based on amino-acid sequences, the protein tertiary structure can be considered additional information. Another example is recognizing objects in images; the textual information in the form of image tags contains additional object descriptions and can be used as privileged.

Theory/Solution

We formalize the LUPI setup for the task of supervised binary classification with a single source of privileged information. Assume that we are given a set of N training examples, represented by feature vectors $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathcal{X} = \mathbb{R}^d$, their label annotation, $Y = \{y_1, \ldots, y_N\} \in \mathcal{Y} = \{+1, -1\}$, and additional information, also in the form of feature vectors, $X^* = \{\mathbf{x}_1^*, \ldots, \mathbf{x}_N^*\} \subset \mathcal{X}^* = \mathbb{R}^{d^*}$, where $\mathbf{x}_i^*$ encodes the additional information we have about sample $\mathbf{x}_i$. We will refer to $\mathcal{X}$ and $\mathcal{X}^*$ as the original and the privileged data spaces, respectively. For learning a binary classification function $f : \mathcal{X} \to \mathcal{Y}$ from a space of all possible functions $\mathcal{F}$, one of the well-established methods is Support Vector Machine (Editor, this is a link to another encyclopedia entry called **Support Vector Machine**.) In Vapnik and Vashist (2009), the SVM+ method was introduced as a generalization of the SVM-based framework to solve LUPI. This SVM+ formulation can be directly extended for multiple sources of privileged information and for the regression estimation problem (more details in Vapnik and Vashist 2009).

SVM+

The SVM+ optimization admits the following form:

$$\underset{\substack{\mathbf{w}\in\mathbb{R}^d, b\in\mathbb{R}\\ \mathbf{w}^*\in\mathbb{R}^{d^*}, b^*\in\mathbb{R}}}{\text{minimize}} \ \frac{1}{2}\left(\|\mathbf{w}\|^2 + \gamma\|\mathbf{w}^*\|^2\right) + C\sum_{i=1}^{N}\langle\mathbf{w}^*, \mathbf{x}_i^*\rangle + b^* \tag{1a}$$

subject to, for all $i = 1, \ldots, N$,

$$y_i[\langle\mathbf{w}, \mathbf{x}_i\rangle + b] \geq 1 - [\langle\mathbf{w}^*, \mathbf{x}_i^*\rangle + b^*] \tag{1b}$$

$$\text{and} \qquad \langle\mathbf{w}^*, \mathbf{x}_i^*\rangle + b^* \geq 0. \tag{1c}$$

The above problem is a generalization of nonlinearly separable (soft-margin) SVM in which the slack variables are parameterized with unknown $\mathbf{w}^*$ and b^*, so that the slack value for each sample is $\xi_i = \langle\mathbf{w}^*, \mathbf{x}_i^*\rangle + b^*$.

The SVM+ optimization problem can be solved in the dual representation using a standard quadratic programming (QP) solver. For a medium-sized problem (thousands to hundred thousands of samples), a general purpose QP solver might not suffice, and special purpose algorithms have to be developed to solve the QP. In Pechyony and Vapnik (2011), suitable sequential minimal optimization (SMO) algorithms were derived to tackle the problem.

Motivation of SVM+

When the number of training examples N increases, soft-margin SVM solutions are known Vapnik (1999) to converge with a rate of $\mathrm{O}\left(\frac{1}{\sqrt{N}}\right)$ to the optimal classifier in the function class $\mathcal{F}$. This is in sharp contrast to hard-margin (linearly separable) solutions that converge with a rate of $\mathrm{O}\left(\frac{1}{N}\right)$. The difference in the learning rate lies in the number of variables to be estimated from the data. In the hard-margin SVM, the weight vector $\mathbf{w}$ and bias parameter b have to be learned. For the soft-margin SVM, in addition to the weight and bias parameters, N slack variables ξ_i – one for each training sample – have to be estimated. The idea of the SVM+ classifier is based on a direct observation that the soft-margin SVM can be turned into the hard-margin SVM if one had access to a so-called *slack oracle* that knows the optimal slack value ξ_i. Since, in practice, we do not have access to the oracle, we can instead estimate the slack function using the additional privileged data.

Relation to Existing Learning Settings

Other than the LUPI framework, there are several machine learning frameworks that exploit multiple sources of information to learn the classification model, among others, domain adaptation, multi-view learning, and multimodal learning. In Table 1, we overview the commonalities and differences of those settings for two sources of information, $\mathcal{X}$ and $\mathcal{X}^*$, and binary classification task $\mathcal{Y} = \{-1, +1\}$.

Applications

The LUPI framework is of interest for practical applications because it captures the typical learning-from-data setting where domain experts are able to provide additional knowledge about the data. For instance, Vapnik and Vashist (2009) explored protein tertiary structures (geometrical shapes) as additional information when the learning task is about classifying proteins based on their secondary structures (amino-acid sequences). In the context of recognizing animal objects such as leopard, panda, or horse in images, Sharmanska et al. (2013) considered physical attributes or properties such as furry, stripes, or smelly associated with the animals to be the extra information. In the longer term, the LUPI framework might have potential applications in the medical settings. For example, in terms of machine learning-aided diagnosis technologies, results from advanced medical tests such as fMRI, CT scans, or X-ray can be used as additional knowledge to compliment routinely collected test results such as the heart rate or the pulse. In the three settings described above, *protein geometrical shapes*, *object physical attributes*, and *advanced medical tests* share the properties that they are very hard and time consuming to gather data and are expensive to provide at test time.

Current and Future Directions

LUPI is an active research area both within machine learning and in application areas such as finance (Ribeiro et al. 2010) and computer vision (Sharmanska et al. 2013; Li et al. 2014). In machine learning, there is interest in algorithms and theoretical aspects of LUPI (Pechyony and Vapnik 2011, 2010; Lapin et al. 2014) and also in adapting SVM-based LUPI to other learning settings. For the latter, LUPI was generalized to learning to rank (Sharmanska et al. 2013), clustering (Feyereisl and Aickelin 2012), metric learning (Fouad et al. 2013), and structured prediction (Feyereisl et al. 2014) and in the Bayesian nonparametric (Editor, this is a link

Learning Using Privileged Information, Table 1 Different machine learning settings according to the availability of the data

Learning setting	Definition	Train domain	Test domain	Illustration
LUPI	Learning with additional information that is only available at training time	$\mathcal{X}, \mathcal{X}^*$	$\mathcal{X}$	$\mathcal{X}$: Images, $\mathcal{X}^*$: Bounding boxes
Domain adaptation	Learning to adapt the classifier from the training domain (also called source) to the test domain (also called target)	$\mathcal{X}$	$\mathcal{X}^*$	$\mathcal{X}$: Amazon images, $\mathcal{X}^*$: Webcam images
Multi-view/multimodal	Learning from multiple feature representations (multiple domains)	$\mathcal{X}, \mathcal{X}^*$	$\mathcal{X}, \mathcal{X}^*$	$\mathcal{X}$: Images, $\mathcal{X}^*$: Image tags

to another encyclopedia entry called **Bayesian nonparametric**.) setting using Gaussian Processes (Editor, this is a link to another encyclopedia entry called **Gaussian Processes**.) (Hernández-Lobato et al. 2014). To compare different types of privileged information, a recent research direction (Wang et al. 2014) explores a bias-variance decomposition (Editor, this is a link to another encyclopedia entry called **bias-variance decomposition**.) tool. The work concludes that a useful privileged information is the one that leads to a large reduction in variance with only a slight penalty in bias with respect to the model trained without privileged information.

Cross-References

- ▶ Inductive Transfer
- ▶ Transfer of Knowledge Across Domains

Recommended Reading

Feyereisl J, Aickelin U (2012) Privileged information for data clustering. Inf Sci 194:4–23

Feyereisl J, Kwak S, Son J, Han B (2014) Object localization based on structural svm using privileged information. In: Neural information processing systems (NIPS), Montreal

Fouad S, Tino P, Raychaudhury S, Schneider P (2013) Incorporating privileged information through metric learning. IEEE Trans Neural Netw Learn Syst 24(7):1086–1098

Hernández-Lobato D, Sharmanska V, Kersting K, Lampert CH, Quadrianto N (2014) Mind the nuisance: Gaussian process classification using privileged noise. In: Neural information processing systems (NIPS), Montreal

Lapin M, Hein M, Schiele B (2014) Learning using privileged information: SVM+ and weighted SVM. Neural Netw 53:95-108

Li W, Niu L, Xu D (2014) Exploiting privileged information from web data for image categorization. In: European conference on computer vision (ECCV), Zurich

Pechyony D, Vapnik V (2010) On the theory of learning with privileged information. In: Neural information processing systems (NIPS), Vancouver

Pechyony D, Vapnik V (2011) Fast optimization algorithms for solving SVM+. In: Statistical Learning and Data Science.

Ribeiro B, Silva C, Vieira A, Gaspar-Cunha A, das Neves J (2010) Financial distress model prediction using SVM+. In: International joint conference on neural networks (IJCNN), Barcelona

Sharmanska V, Quadrianto N, Lampert CH (2013) Learning to rank using privileged information. In: International conference on computer vision (ICCV), Sydney

Vapnik V (1999) The nature of statistical learning theory. Springer, Berlin/Heidelberg

Vapnik V, Vashist A (2009) A new learning paradigm: learning using privileged information. Neural Netw 22(5):544–557

Wang Z, Wang X, Ji Q (2014) Learning with hidden information. In: International Conference on Pattern Recognition (ICPR), Stockholm

Learning Vector Quantization

Synonyms

LVQ

Definition

Learning vector quantization (LVQ) algorithms produce prototype-based classifiers. Given a set of labeled prototype vectors, each input vector is mapped to the closest prototype, and classified according to its label. The basic LVQ learning algorithm works by iteratively moving the closest prototype toward the current input if their labels are the same, and away from the input if not. Some variants of the algorithm have been shown to approximate Bayes optimal decision borders. The algorithm was introduced by Kohonen, and being prototype-based it bears close resemblance to ▶ competitive learning and ▶ Self-Organizing Maps. The differences are that LVQ is supervised and the prototypes are not ordered (i.e., there is no neighborhood function).

Learning with Different Classification Costs

▶ Cost-Sensitive Learning

Learning with Hidden Context

▶ Concept Drift

Learning Word Senses

▶ Word Sense Disambiguation

Least-Squares Reinforcement Learning Methods

Michail G. Lagoudakis
Technical University of Crete, Chania, Greece

Abstract

Most algorithms for sequential decision making rely on computing or learning a value function that captures the expected long-term return of a decision at any given state. Value functions are in general complex, nonlinear functions that cannot be represented compactly as they are defined over the entire state or state-action space. Therefore, most practical algorithms rely on value function approximation methods, and the most common choice for approximation architecture is a linear architecture. Exploiting the properties of linear architectures, a number of efficient learning algorithms based on least-squares techniques have been developed. These algorithms focus on different aspects of the approximation problem and deliver diverse solutions; nevertheless they share the tendency to process data collectively (batch mode) and, in general, achieve better results compared to their counterpart algorithms based on stochastic approximation.

Definition

Least-Squares Reinforcement Learning Methods are methods that focus on the problem of reinforcement learning (learning by trial and error) using least-squares techniques (optimization techniques for deriving solutions that minimize some form of squared error measure).

Motivation and Background

Consider a Markov decision process (MDP) $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mathcal{D})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}(s'|s,a)$ is a Markovian transition model, $\mathcal{R}(s,a)$ is a reward function, $\gamma \in (0,1]$ is the discount factor, and $\mathcal{D}$ is the initial state distribution. A linear approximation architecture approximates the value function $V^\pi(s)$ or $Q^\pi(s,a)$ of a stationary (stochastic) policy $\pi(a|s)$ as a linear weighted combination of linearly independent basis functions or features ϕ:

$$\hat{V}^\pi(s;w) = \sum_{j=1}^{k} \phi_j(s) w_j = \phi(s)^\top w$$

$$\hat{Q}^\pi(s,a;w) = \sum_{j=1}^{m} \phi_j(s,a) w_j = \phi(s,a)^\top w.$$

The parameters or weights of the approximation are the coefficients w.

Let V^π and $\hat{V}^\pi$ be the exact and the approximate, respectively, state value function of a policy π, both given as column vectors of size $|\mathcal{S}|$. Define $\boldsymbol{\Phi}_V$ as the $(|\mathcal{S}| \times k)$ matrix with elements $\phi_j(s)$, where $s \in \mathcal{S}$ span the rows and $j = 1, 2, \ldots, k$ span the columns. Then, $\hat{V}^\pi$ can be expressed compactly as $\hat{V}^\pi = \boldsymbol{\Phi}_V w^\pi$. Similarly, let Q^π and $\hat{Q}^\pi$ be the exact and the approximate, respectively, state-action value function of a policy π, both given as column vectors of size $|\mathcal{S}||A|$. Define $\boldsymbol{\Phi}_Q$ as the $(|\mathcal{S}||\mathcal{A}| \times m)$ matrix with elements $\phi_j(s,a)$, where $(s,a) \in (\mathcal{S} \times \mathcal{A})$ span the rows and $j = 1, 2, \ldots, m$ span the columns. Then, $\hat{Q}^\pi$ can be expressed compactly as $\hat{Q}^\pi = \boldsymbol{\Phi}_Q w^\pi$. In addition, let $\mathcal{R}$ be a vector of size $|\mathcal{S}||\mathcal{A}|$ with entries $\mathcal{R}(s,a)$ that contains the reward function, $\mathbf{P}$ be a stochastic matrix of size $(|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}|)$ that contains the transition model $(\mathbf{P}((s,a),s') = \mathcal{P}(s'|s,a))$, and $\boldsymbol{\Pi}_\pi$ be a stochastic matrix of size $(|\mathcal{S}| \times |\mathcal{S}||\mathcal{A}|)$ that describes policy π $(\boldsymbol{\Pi}_\pi(s,(s,a)) = \pi(a|s))$. The state value function V^π and the state-action value function Q^π are the solutions of the linear Bellman equations

$$\begin{aligned} V^\pi &= \mathbf{\Pi}_\pi(\mathcal{R} + \gamma \mathbf{P} V^\pi) \\ Q^\pi &= \mathcal{R} + \gamma \mathbf{P} \mathbf{\Pi}_\pi Q^\pi \end{aligned}$$

and also the fixed points of the corresponding linear Bellman operators

$$\begin{aligned} V^\pi &= T_V^\pi(V^\pi), \text{ where } T_V^\pi(x) = \mathbf{\Pi}_\pi(\mathcal{R} + \gamma \mathbf{P} x) \\ Q^\pi &= T_Q^\pi(Q^\pi), \text{ where } T_Q^\pi(x) = \mathcal{R} + \gamma \mathbf{P} \mathbf{\Pi}_\pi x. \end{aligned}$$

If V^π and Q^π were known, they could be projected orthogonally onto the space spanned by the basis functions to obtain the optimal least-squares approximation. (For simplicity of presentation, we consider only uniform least-squares criteria in this text, but generalization to weighted least-squares criteria is possible in all cases.) For the state value function, we have

$$\begin{aligned} \hat{V}^\pi &= \boldsymbol{\Phi}_V w^\pi \\ &= \boldsymbol{\Phi}_V \left(\boldsymbol{\Phi}_V^\top \boldsymbol{\Phi}_V\right)^{-1} \boldsymbol{\Phi}_V^\top V^\pi \implies w^\pi \\ &= \boldsymbol{\Phi}_V^{-1} \boldsymbol{\Phi}_V \left(\boldsymbol{\Phi}_V^\top \boldsymbol{\Phi}_V\right)^{-1} \boldsymbol{\Phi}_V^\top V^\pi, \end{aligned}$$

whereas for the state-action value function, we have

$$\begin{aligned} \hat{Q}^\pi &= \boldsymbol{\Phi}_Q w^\pi \\ &= \boldsymbol{\Phi}_Q \left(\boldsymbol{\Phi}_Q^\top \boldsymbol{\Phi}_Q\right)^{-1} \boldsymbol{\Phi}_Q^\top Q^\pi \implies w^\pi \\ &= \boldsymbol{\Phi}_Q^{-1} \boldsymbol{\Phi}_Q \left(\boldsymbol{\Phi}_Q^\top \boldsymbol{\Phi}_Q\right)^{-1} \boldsymbol{\Phi}_Q^\top Q^\pi. \end{aligned}$$

The learning algorithms described here strive to find a set of parameters w, such that the approximate value function is a good approximation to the true one. However, since the exact value functions are unknown, these algorithms have to rely on information contained in the Bellman equations and the Bellman operators to derive expressions that characterize a good choice for w. It has been shown that, in many cases, this kind of learning is equivalent to approximating the MDP using a linear (compressed) model and solving exactly the approximate model (Parr et al. 2008).

Bellman Residual Minimizing Approximation

An obvious approach to deriving a good approximation is to require that the approximate function satisfies the linear Bellman equation as closely as possible. Substituting the approximation $\hat{V}^\pi$ into the Bellman equation for V^π yields an overconstrained linear system over the k parameters w^π:

$$\begin{aligned} \hat{V}^\pi &\approx \mathbf{\Pi}_\pi \left(\mathcal{R} + \gamma \mathbf{P} \hat{V}^\pi\right) \\ \boldsymbol{\Phi}_V w^\pi &\approx \mathbf{\Pi}_\pi \left(\mathcal{R} + \gamma \mathbf{P} \boldsymbol{\Phi}_V w^\pi\right) \\ \left(\boldsymbol{\Phi}_V - \gamma \mathbf{\Pi}_\pi \mathbf{P} \boldsymbol{\Phi}_V\right) w^\pi &\approx \mathbf{\Pi}_\pi \mathcal{R}. \end{aligned}$$

Solving this overconstrained system in the least-squares sense is a $(k \times k)$ system

$$\begin{aligned} &\left(\boldsymbol{\Phi}_V - \gamma \mathbf{\Pi}_\pi \mathbf{P} \boldsymbol{\Phi}_V\right)^\top \left(\boldsymbol{\Phi}_V - \gamma \mathbf{\Pi}_\pi \mathbf{P} \boldsymbol{\Phi}_V\right) w^\pi \\ &\quad = \left(\boldsymbol{\Phi}_V - \gamma \mathbf{\Pi}_\pi \mathbf{P} \boldsymbol{\Phi}_V\right)^\top \mathbf{\Pi}_\pi \mathcal{R} \end{aligned} \tag{1}$$

whose solution is unique and minimizes $\|T_V^\pi\left(\hat{V}^\pi\right) - \hat{V}^\pi\|_2$. Similarly, substituting the approximation $\hat{Q}^\pi$ into the Bellman equation for Q^π yields an overconstrained linear system over the m parameters w^π:

$$\begin{aligned} \hat{Q}^\pi &\approx \mathcal{R} + \gamma \mathbf{P} \mathbf{\Pi}_\pi \hat{Q}^\pi \\ \boldsymbol{\Phi}_Q w^\pi &\approx \mathcal{R} + \gamma \mathbf{P} \mathbf{\Pi}_\pi \boldsymbol{\Phi}_Q w^\pi \\ \left(\boldsymbol{\Phi}_Q - \gamma \mathbf{P} \mathbf{\Pi}_\pi \boldsymbol{\Phi}_Q\right) w^\pi &\approx \mathcal{R}. \end{aligned}$$

Solving this overconstrained system in the least-squares sense is a $(m \times m)$ system

$$\begin{aligned} &\left(\boldsymbol{\Phi}_Q - \gamma \mathbf{P} \mathbf{\Pi}_\pi \boldsymbol{\Phi}_Q\right)^\top \left(\boldsymbol{\Phi}_Q - \gamma \mathbf{P} \mathbf{\Pi}_\pi \boldsymbol{\Phi}_Q\right) w^\pi \\ &\quad = \left(\boldsymbol{\Phi}_Q - \gamma \mathbf{P} \mathbf{\Pi}_\pi \boldsymbol{\Phi}_Q\right)^\top \mathcal{R} \end{aligned} \tag{2}$$

whose solution is unique and minimizes $\|T_Q^\pi\left(\hat{Q}^\pi\right) - \hat{Q}^\pi\|_2$. In both cases, the solution minimizes the L_2 norm of the Bellman residual (the difference between the left-hand side and the right-hand side of the linear Bellman equation).

Least-Squares Fixed-Point Approximation

Recall that a value function is also the fixed point of the corresponding linear Bellman operator. Another way to go about finding a good approximation is to force the approximate value function to be a fixed point under the linear Bellman operator. For that to be possible, the fixed point has to lie in the space of approximate value functions which is the space spanned by the basis functions. Even though the approximate function itself lies in that space by definition, the result of applying the linear Bellman operator to the approximation will in general be out of that space and must be projected back. Considering the orthogonal projection $\left(\boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\right)$ (which minimizes the L_2 norm) onto the column space of $\boldsymbol{\Phi}$, we seek an approximate value function that is invariant under one application of the linear Bellman operator followed by orthogonal projection onto the space spanned by the basis functions. More specifically, for the state value function, we require that

$$\begin{aligned}\hat{V}^\pi &= \boldsymbol{\Phi}_V\left(\boldsymbol{\Phi}_V^\top\boldsymbol{\Phi}_V\right)^{-1}\boldsymbol{\Phi}_V^\top\left(T_V^\pi(\hat{V}^\pi)\right)\\ \boldsymbol{\Phi}_V w^\pi &= \boldsymbol{\Phi}_V\left(\boldsymbol{\Phi}_V^\top\boldsymbol{\Phi}_V\right)^{-1}\\ &\qquad\boldsymbol{\Phi}_V^\top\left(\boldsymbol{\Pi}_\pi(\mathcal{R}+\gamma\mathbf{P}\boldsymbol{\Phi}_V w^\pi)\right).\end{aligned}$$

Note that the orthogonal projection to the column space of $\boldsymbol{\Phi}_V$ is well defined, because the columns of $\boldsymbol{\Phi}_V$ (the basis functions) are linearly independent by definition. The expression above is equivalent to solving a $(k \times k)$ linear system

$$\boldsymbol{\Phi}_V^\top(\boldsymbol{\Phi}_V - \gamma\boldsymbol{\Pi}_\pi\mathbf{P}\boldsymbol{\Phi}_V)w^\pi = \boldsymbol{\Phi}_V^\top\boldsymbol{\Pi}_\pi\mathcal{R} \qquad (3)$$

whose solution is guaranteed to exist for all, but finitely many, values of γ (Koller and Parr 2000) and minimizes (in fact, zeros out) the projected Bellman residual. Since the orthogonal projection minimizes the L_2 norm, the solution w^π yields a value function $\hat{V}^\pi$ which is the least-squares fixed-point approximation to the true state value function. Similarly, for the state-action value function, we require that

$$\begin{aligned}\hat{Q}^\pi &= \boldsymbol{\Phi}_Q\left(\boldsymbol{\Phi}_Q^\top\boldsymbol{\Phi}_Q\right)^{-1}\boldsymbol{\Phi}_Q^\top\left(T_Q^\pi(\hat{Q}^\pi)\right)\\ \boldsymbol{\Phi}_Q w^\pi &= \boldsymbol{\Phi}_Q\left(\boldsymbol{\Phi}_Q^\top\boldsymbol{\Phi}_Q\right)^{-1}\\ &\qquad\boldsymbol{\Phi}_Q^\top\left(\mathcal{R}+\gamma\mathbf{P}\boldsymbol{\Pi}_\pi\boldsymbol{\Phi}_Q w^\pi\right).\end{aligned}$$

This is equivalent to solving a $(m \times m)$ linear system

$$\boldsymbol{\Phi}_Q^\top\left(\boldsymbol{\Phi}_Q - \gamma\mathbf{P}\boldsymbol{\Pi}_\pi\boldsymbol{\Phi}_Q\right)w^\pi = \boldsymbol{\Phi}_Q^\top\mathcal{R} \qquad (4)$$

whose solution is again guaranteed to exist for all, but finitely many, values of γ (Koller and Parr 2000) and minimizes (in fact, zeros out) the projected Bellman residual. Since the orthogonal projection minimizes the L_2 norm, the solution w^π yields a value function $\hat{Q}^\pi$ which is the least-squares fixed-point approximation to the true state-action value function.

Structure of Learning System

Least-Squares Temporal Difference Learning

The least-squares temporal difference (LSTD) learning algorithm (Bradtke and Barto 1996) learns the least-squares fixed-point approximation to the state value function V^π of a fixed policy π. In essence, LSTD attempts to form and solve the linear system of Eq. 3 using sampling. Each sample (s, r, s') indicates a minimal interaction with the unknown process, whereby in some state s, a decision was made using policy π, and reward r was observed, as well as a transition to state s'. LSTD processes a set of samples collectively to derive the weights of the approximate value function. LSTD is an on-policy method; it requires that all training samples are collected using the policy under evaluation. The LSTD algorithm is summarized in Algorithm 1.

LSTD improves upon the temporal difference (TD) learning algorithm for linear architectures by making efficient use of data and converging

Algorithm 1 Least-squares temporal difference (LSTD)

```
$w = \textbf{LSTD}(D, k, \phi, \gamma)$
  Input: samples $D$, integer $k$, basis functions $\phi$, discount factor $\gamma$
  Output: weights $w$ of the learned state value function

  $\mathbf{A} \leftarrow \mathbf{0}$          // $(k \times k)$ matrix
  $b \leftarrow \mathbf{0}$          // $(k \times 1)$ vector
  for each sample $(s, r, s') \in D$ do
    $\mathbf{A} \leftarrow \mathbf{A} + \phi(s)\Big(\phi(s) - \gamma\phi(s')\Big)^\top$
    $b \leftarrow b + \phi(s)r$
  end for
  $w \leftarrow \mathbf{A}^{-1}b$
  return $w$
```

Algorithm 2 Bellman residual minimization learning (BRML)

```
$w = \textbf{BRML}(D, k, \phi, \gamma)$
  Input: paired samples $D$, integer $k$, basis functions $\phi$, discount factor $\gamma$
  Output: weights $w$ of the learned state value function

  $\mathbf{A} \leftarrow \mathbf{0}$          // $(k \times k)$ matrix
  $b \leftarrow \mathbf{0}$          // $(k \times 1)$ vector
  for each pair of samples $[(s, r, s'), (s, r, s'')] \in D$ do
    $\mathbf{A} \leftarrow \mathbf{A} + \Big(\phi(s) - \gamma\phi(s')\Big)\Big(\phi(s) - \gamma\phi(s'')\Big)^\top$
    $b \leftarrow b + \Big(\phi(s) - \gamma\phi(s')\Big)r$
  end for
  $w \leftarrow \mathbf{A}^{-1}b$
  return $w$
```

faster. The main advantage of LSTD over TD is the elimination of the slow stochastic approximation and the learning rate that needs careful adjustment. TD uses samples to make small modifications and then discards them. In contrast, with LSTD, the information gathered from a single sample remains present in the matrices of the linear system and is used in full every time the parameters are computed. In addition, as a consequence of the elimination of stochastic approximation, LSTD does not diverge.

LSTD(λ) (Boyan 1999) is an extension to LSTD that spans the spectrum between LSTD ($\lambda = 0$) and linear regression over Monte Carlo returns ($\lambda = 1$) for $\lambda \in [0, 1]$. LSTD(λ) for $\lambda > 0$ requires that samples come from complete episodes. RLSTD(λ) is a variant of LSTD(λ) that uses recursive least-squares techniques for efficient implementation (Xu et al. 2002).

Bellman Residual Minimization Learning

The main idea behind LSTD can also be used to learn the Bellman residual minimizing approximation to the state value function V^π of a fixedpolicy π. In this case, the goal is to form and solve the linear system of Eq. 1 using sampling. However, the structure of the system, in this case, requires that samples are "paired," which means that two independent samples (s, r, s') and (s, r, s'') for the same state s must be drawn to perform one update. This is necessary to obtain unbiased estimates of the system matrices. Each sample (s, r, s') again indicates a minimal interaction with the unknown process, whereby in some state s, a decision was made using policy π, and reward r was observed, as well as a transition to state s'. Obtaining paired samples is trivial with a generative model (a simulator) of the process, but virtually impossible when samples are drawn directly from a physical process. This fact makes the Bellman residual minimizing approximation somewhat impractical for learning, but otherwise a reasonable approach for computing approximate state value functions from the model of the process (Schweitzer and Seidmann 1985). The learning algorithm for Bellman residual minimization is summarized in Algorithm 2.

Hybrid Least-Squares Learning

Value function learning algorithms, either in the Bellman residual minimization or in the fixed-point sense, have been used within approximate policy iteration schemes for policy learning, but in practice, they exhibit quite diverse performance. Fixed-point approximations tend to deliver better policies, whereas Bellman residual minimizing approximations fluctuate less between different rounds of policy iteration. Motivated by a desire to combine the advantages of both approximations, some researchers have focused on learning hybrid approximations that lie somewhere between these two extremes. Johns et al. (2009) have proposed two different approaches to combining these two approximations. The first relies on a derivation that begins with the goal of minimizing a convex combination of the two objectives (Bellman residual and projected Bellman residual); the resulting learning algorithm is quite expensive as it requires the maintenance of three matrices and two vectors (as opposed to one matrix and one vector when learning a non-hybrid approximation), as well as the inversion of one of the three matrices at each update. The second approach focuses directly on a convex combination of the linear systems produced by the two extreme approximations (Eqs. 1 and 3); the resulting learning algorithm has the same complexity as non-hybrid algorithms and in many cases exhibits better performance than the original approximations. On the other hand, both hybrid learning algorithms still have to deal with the paired samples problem and additionally require tuning of the convex combination parameter.

Least-Squares Policy Evaluation

The least-squares policy evaluation (LSPE) learning algorithm (Nedić and Bertsekas 2003), like LSTD, learns the least-squares fixed-point approximation to the state value function V^{π} of a fixed policy π. Both LSPE and LSTD strive to obtain the solution to the same linear system (Eq. 3) but using different methods; LSPE uses an iterative method, whereas LSTD uses direct matrix inversion. Unlike LSTD, LSPE begins with some arbitrary approximation to the value function (given by a parameter vector w') and focuses on the one-step application of the Bellman operator within the lower-dimensional space spanned by the basis functions aiming at producing an incremental improvement on the parameters. In that sense, LSPE can take advantage of a good initialization of the parameter vector. Given the current parameters w' and a set $\{(s_k, r_k, s'_k) : k = 0, \ldots, t\}$ of samples, LSPE first computes the solution $\bar{w}$ to the least-squares problem

$$\min_{w} \sum_{k=0}^{t} \Big(\phi(s_k)^{\top} w - \big(r_k + \gamma \phi(s'_k)^{\top} w'\big)\Big)^2$$

Algorithm 3 Least-squares policy evaluation (LSPE)

```
$w = \mathbf{LSPE}(D, k, \phi, \gamma, w', \alpha)$
  Input: samples $D$, integer $k$, features $\phi$, discount factor $\gamma$, weights $w'$, stepsize $\alpha$
  Output: weights $w$ of the learned state value function

  $\mathbf{A} \leftarrow \mathbf{0}$            // $(k \times k)$ matrix
  $b \leftarrow \mathbf{0}$             // $(k \times 1)$ vector
  for each sample $(s, r, s') \in D$ do
      $\mathbf{A} \leftarrow \mathbf{A} + \phi(s)\phi(s)^{\top}$
      $b \leftarrow b + \phi(s)\big(r + \gamma\phi(s')^{\top} w'\big)$
  end for
  $\bar{w} \leftarrow \mathbf{A}^{-1} b$
  $w \leftarrow \alpha w' + (1 - \alpha)\bar{w}$
  return $w$
```

and then updates w' toward $\bar{w}$ using a stepsize $\alpha \in (0, 1]$. The LSPE algorithm is summarized in Algorithm 3.

The LSPE incremental update at the extreme can be performed whenever a new sample arrives or whenever a batch of samples becomes available to remedy computational costs. An efficiency improvement to LSPE is to use recursive least-squares computations, so that the least-squares problem can be solved without matrix inversion. LSPE(λ) for $\lambda \in [0, 1]$ is an extension of LSPE to multistep updates in the same spirit as LSTD(λ). LSPE(λ) for $\lambda > 0$ requires that samples come from complete episodes.

Least-Squares Policy Iteration

Least-squares policy iteration (LSPI) (Lagoudakis and Parr 2003) is a model-free, reinforcement learning algorithm for policy learning based on the approximate policy iteration framework. LSPI learns in a batch manner by processing multiple times the same set of samples. LSPI is an off-policy method; samples can be collected arbitrarily from the process using any policy. Each sample (s, a, r, s') indicates that the learner observed the current state s, chose an action a, and observed the resulting next state s' and the reward received r. LSPI iteratively learns a (weighted) least-squares fixed-point approximation of the state-action value functions (Eq. 4) of a sequence of improving (deterministic) policies π. At each iteration, the

Algorithm 4 Least-squares policy iteration (LSPI)

```
$w$ = LSPI($D, m, \phi, \gamma, \epsilon$)
  Input: samples $D$, integer $m$, basis functions $\phi$, discount factor $\gamma$, tolerance $\epsilon$
  Output: weights $w$ of the learned value function of the best learned policy

  $w \leftarrow \mathbf{0}$
  repeat
    $\mathbf{A} \leftarrow \mathbf{0}$        // $(m \times m)$ matrix
    $b \leftarrow \mathbf{0}$         // $(m \times 1)$ vector
    $w' \leftarrow w$
    for each sample $(s, a, r, s')$ in $D$ do
      $a' = \arg\max_{a'' \in \mathcal{A}} \phi(s', a'')^\top w'$
      $\mathbf{A} \leftarrow \mathbf{A} + \phi(s, a)\Big(\phi(s, a) - \gamma\phi(s', a')\Big)^\top$
      $b \leftarrow b + \phi(s, a)r$
    end for
    $w \leftarrow \mathbf{A}^{-1}b$
  until $(\|w - w'\| < \epsilon)$
  return $w$
```

Algorithm 5 Least-squares fitted Q-iteration

```
$w$ = LS-FQI($D, m, \phi, \gamma, N$)
  Input: samples $D$, integer $m$, basis functions $\phi$, discount factor $\gamma$, iterations $N$
  Output: weights $w$ of the learned value function of the best learned policy

  $i \leftarrow 0$
  $w \leftarrow \mathbf{0}$
  while $(i < N)$ do
    $\mathbf{A} \leftarrow \mathbf{0}$        // $(m \times m)$ matrix
    $b \leftarrow \mathbf{0}$         // $(m \times 1)$ vector
    for each sample $(s, a, r, s')$ in $D$ do
      $\mathbf{A} \leftarrow \mathbf{A} + \phi(s, a)\phi(s, a)^\top$
      $b \leftarrow b + \phi(s, a)\Big(r + \gamma \max_{a' \in \mathcal{A}}$
        $\{\phi(s', a')^\top w\}\Big)$
    end for
    $w \leftarrow \mathbf{A}^{-1}b$
    $i \leftarrow i + 1$
  end while
  return $w$
```

value function of the policy is approximated by solving a $(m \times m)$ linear system, formed using the single sample set and the policy from the previous iteration. LSPI offers a non-divergence guarantee, and in most cases, it converges in just a few iterations. LSPI exhibits excellent sample efficiency and has been used widely in many domains. Algorithm 4 summarizes LSPI.

The default internal policy evaluation procedure in LSPI is the variation of LSTD for the state-action value function (LSTDQ). However, any other value function learning algorithm, such as BRML or LSPE, could be used instead; nevertheless, the λ extensions are not applicable in this case, because the samples in LSPI have been collected arbitrarily and not by the policy being evaluated each time. The variation of LSPI that internally learns the Bellman residual minimizing approximation (Eq. 2) using BRML has produced inferior policies, in general, and suffers from the paired samples problem.

Least-Squares Fitted Q-Iteration

Fitted Q-iteration (FQI) (Ernst et al. 2005) is a batch reinforcement learning algorithm for policy learning based on the popular Q-learning algorithm. FQI uses an iterative scheme to approximate the optimal value function, whereby an improved value function Q is learned at each iteration by fitting a function approximator to a set of training examples generated using a set of samples from the process and the Q-learning update rule. Any function approximation architecture and the corresponding supervised learning algorithm could be used in the iteration. The simplest choice is to use least-squares regression along with a linear architecture to learn the least-squares fixed-point approximation of the state-action value function (Eq. 4). This version of least-squares fitted Q-iteration is summarized in Algorithm 5. In a sense, this version of FQI combines ideas from LSPE and LSPI. Like LSPI, FQI is an off-policy method; samples can be collected arbitrarily from the process using any policy. In practice, FQI produces very good policies within a moderate number of iterations.

Cross-References

- ▶ Curse of Dimensionality
- ▶ Feature Selection
- ▶ Radial Basis Function Approximation
- ▶ Reinforcement Learning
- ▶ Temporal Difference Learning

Recommended Reading

Boyan JA (1999) Least-squares temporal difference learning. In: Proceedings of the sixteenth international conference on machine learning, Bled, pp 49–56

Bradtke SJ, Barto AG (1996) Linear least-squares algorithms for temporal difference learning. Mach Learn 22:33–57

Ernst D, Geurts P, Wehenkel L (2005) Tree-based batch mode reinforcement learning. J Mach Learn Res 6:503–556

Johns J, Petrik M, Mahadevan S (2009) Hybrid least-squares algorithms for approximate policy evaluation. Mach Learn 76(2–3):243–256

Koller D, Parr R (2000) Policy iteration for factored MDPs. In: Proceedings of the sixteenth conference on uncertainty in artificial intelligence, Stanford, pp 326–334

Lagoudakis MG, Parr R (2003) Least-squares policy iteration. J Mach Learn Res 4:1107–1149

Nedić A, Bertsekas DP (2003) Least-squares policy evaluation algorithms with linear function approximation. Discret Event Dyn Syst Theory Appl 13(1–2):79–110

Parr R, Li L, Taylor G, Painter-Wakefield C, Littman ML (2008) An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In: Proceedings of the twenty-fifth international conference on machine learning, Helsinki, pp 752–759

Schweitzer PJ, Seidmann A (1985) Generalized polynomial approximations in Markovian decision processes. J Math Anal Appl 110(6):568–582

Xu X, He H-G, Hu D (2002) Efficient reinforcement learning using recursive least-squares methods. J Artif Intell Res 16:259–292

Leave-One-Out Cross-Validation

Definition

Leave-one-out cross-validation is a special case of ▶ cross-validation where the number of folds equals the number of ▶ instances in the ▶ data set. Thus, the learning algorithm is applied once for each instance, using all other instances as a ▶ training set and using the selected instance as a single-item ▶ test set. This process is closely related to the statistical method of jack-knife estimation (Efron 1982).

Cross-References

▶ Algorithm Evaluation

Recommended Reading

Efron B (1982) The Jackknife, the bootstrap and other resampling plans. In: CBMS-NSF regional conference series in applied mathematics 1982. Society for Industrial and Applied Mathematics (SIAM), Philadelphia

Leave-One-Out Error

Synonyms

Hold-one-out error; LOO error

Definition

Leave-one-out error is an estimate of ▶ error obtained by ▶ leave-one-out cross-validation.

Lessons-Learned Systems

▶ Case-Based Reasoning

Lifelong Learning

▶ Cumulative Learning

Life-Long Learning

▶ Continual Learning

Lift

Lift is a measure of the relative utility of a ▶ classification rule. It is calculated by dividing the probability of the consequent of the rule, given its antecedent by the prior probability of the consequent:

$$\text{lift}(x \to y) = P(Y = y|X = x)/P(Y = y).$$

In practice, the probabilities are usually estimated from either ▸ training data or ▸ test data. In this case,

$$\text{lift}(x \to y) = F(Y = y|X = x)/F(Y = y).$$

where $F(Y = y|X = x)$ is the frequency with which the consequent occurs in the data in the context of the antecedent and F($Y = y$) is the frequency of the consequent in the data.

Linear Discriminant

Novi Quadrianto[1] and Wray L. Buntine[2,3]
[1]Department of Informatics, SMiLe CLiNiC, University of Sussex, Brighton, UK
[2]Statistical Machine Learning Program, NICTA, Canberra, ACT, Australia
[3]Faculty of Information Technology, Monash University, Clayton, VIC, Australia

Definition

A discriminant is a function that takes an input variable x and outputs a class label y for it. A linear discriminant is a discriminant that uses a linear function of the input variables and more generally a linear function of some vector function of the input variables $f(x)$.

This entry focuses on one such linear discriminant function called Fisher's linear discriminant. Fisher's discriminant works by finding a projection of input variables to a lower dimensional space while maintaining a class separability property.

Motivation and Background

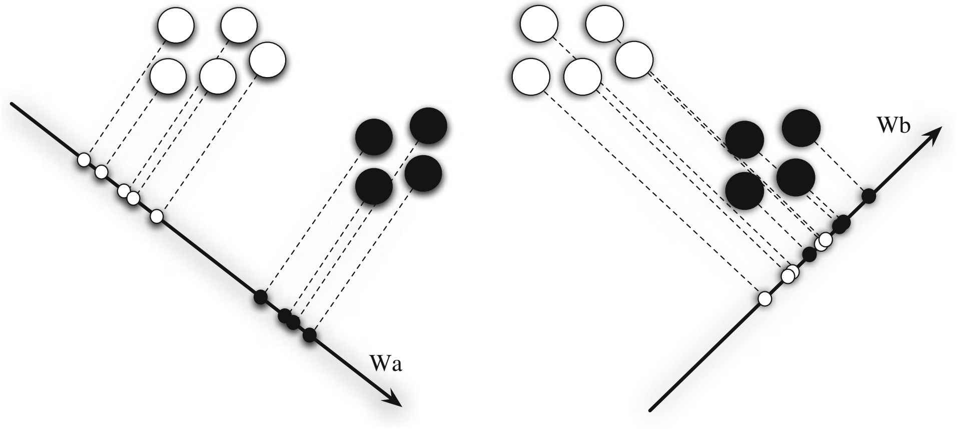
The curse of dimensionality (▸ Curse of Dimensionality) is an ongoing problem in applying statistical techniques to pattern recognition problems. Techniques that are computationally tractable in low-dimensional spaces can become completely impractical in high-dimensional spaces. Consequently, various methods have been proposed to reduce the dimensionality of the input or feature space in the hope of obtaining a more manageable problem. This relies on the fact that real data will often be confined to a region of the space having lower effective dimensionality, and in particular the directions over which important variations in the output variables occur may be so confined. For example, we can reduce a d-dimensional problem to one dimension if we project the d-dimensional data onto a line. However, arbitrary projections will usually produce cluttered projected samples from all of the classes. Thus, the aim is to find a good projection so that the projected samples are well separated. This is exactly the goal of Fisher's linear discriminant analysis.

Fisher's Discriminant for Two-Category Problem

Given N observed training data points $\{(x_i, y_i)\}_{i=1}^N$ where $y_i \in \{1, \ldots, \Omega\}$ is the label for an input variable $x_i \in \mathbb{R}^d$, our task is to find the underlying discriminant function, $f : \mathbb{R}^d \to \{1, \ldots, \Omega\}$. The linear discriminant seeks a projection of d-dimensional input onto a line in the direction of $w \in \mathbb{R}^d$, such that

$$f(x) = w^T x. \quad (1)$$

Subsequently, a class label y can be assigned by thresholding the projected values, for example, for $f(x) \geq C$ we assign $y = 1$ and otherwise we assign $y = 2$ for an appropriate choice of constant C. While the magnitude of w has no real significance (acts only as a scaling factor to y), the direction of w plays a crucial role.



Linear Discriminant, Fig. 1 Black and white encode class labels. Projection of samples onto two different lines. The *plot* on the left shows greater separation between the *white* and *black projected points*

Inappropriate choice of w can result in a non-informative heavily cluttered line. However, by adjusting the components of weight w, we can find a projection that maximizes the class separability (Fig. 1). It is crucial to note that whenever the underlying data distributions are multimodal and highly overlapping, it might not be possible to find such a projection.

Consider a two-category problem, a class label Ω_1 and a class label Ω_2 with N_1 and N_2 number of data points, respectively. The d-dimensional per-class sample mean is given by

$$\mu_1 = \frac{1}{N_1}\sum_{i\in\Omega_1} x_i \qquad \mu_2 = \frac{1}{N_2}\sum_{i\in\Omega_2} x_i. \tag{2}$$

The simplest class separability criterion is the separation of the projected class mean, that is, we can find the weight vector w that maximizes

$$\begin{aligned} m_2 - m_1 &= \frac{1}{N_2}\sum_{i\in\Omega_2} w^T x_i - \frac{1}{N_1}\sum_{i\in\Omega_1} w^T x_i \\ &= w^T(\mu_2 - \mu_1), \end{aligned} \tag{3}$$

where m_1 and m_2 are the projected class means. An additional unit length constraint on w, i.e., $\sum_i w_i^2 = 1$ should be imposed to have a well-defined maximization problem. The above separability criterion produces a line that is parallel to the line joining the two means. However, this projection is suboptimal whenever the data has distinct covariances depending on class (i.e., it is un-isotropic).

Fisher's criterion maximizes a large separation between the projected class means *while simultaneously* minimizing a variance within each class. This criterion can be expressed as

$$J(w) = \frac{w^T S_B w}{w^T S_W w}. \tag{4}$$

where the total within-class covariance matrix is

$$\begin{aligned} S_W &= \sum_{i\in\Omega_1} (x_i - \mu_1)(x_i - \mu_1)^T \\ &\quad + \sum_{i\in\Omega_2} (x_i - \mu_2)(x_i - \mu_2)^T , \end{aligned} \tag{5}$$

and a between-class covariance matrix is

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T . \tag{6}$$

The maximizer of (4) can be found by setting its first derivative with respect to the weights vector to zero, that is,

$$(w^T S_B w) S_W w = (w^T S_W w) S_B w. \tag{7}$$

It is clear from (6), that $S_B w$ admits the direction of $(\mu_2 - \mu_1)$ (Bishop 2006). As only the direction of w is important, we can drop the scaling factors in (7), those are $(w^T S_B w)$ and $(w^T S_W w)$. Multiplying both sides of (7) by S_W^{-1}, we can then obtain the solution of w that maximizes (4) as

$$w = S_W^{-1}(\mu_2 - \mu_1). \tag{8}$$

Fisher's Discriminant for Multi-category Problem

For the general Ω-class problem, we seek a projection from d-dimensional space to a $(\Omega-1)$-dimensional space which is accomplished by $\Omega - 1$ linear discriminant functions, that is,

$$f_c(x) = w_c^T x \qquad c = 1, \ldots, \Omega - 1. \tag{9}$$

In the matrix notation, $f(x) = W^T x$ for $W \in \mathbb{R}^{d\times(\Omega-1)}$ and $f(x) \in \mathbb{R}^{(\Omega-1)}$. The generalization of the within-class covariance matrix in (5) to the case of Ω classes is simply the total within-class covariance matrix over Ω classes, that is $S_W = \sum_{c=1}^{\Omega} S_c$ with $S_c = \sum_{i\in c}(x_i - \mu_c)(x_i - \mu_c)^T$. Following Duda and Hart (1973) and Bishop (2006), the between-class covariance matrix S_B is defined as substracting the within-class covariance matrix from the so-called total covariance matrix, $\sum_{i=1}^{N}(x_i - \mu)(x_i - \mu)^T$ with μ denoting the total sample mean of the dataset. One of the criteria to be optimized is (Fukunaga 1990)

$$J(w) = \mathrm{Trace}((W^T S_W W)^{-1}(W^T S_B W)). \tag{10}$$

The maximizer of (10) is eigenvectors of $S_W^{-1} S_B$ associated with $\Omega - 1$ largest eigenvalues. It is important to note that the between-class covariance matrix S_B is the sum of Ω matrices of rank one or less, and because only $\Omega - 1$ of these matrices are independent, S_B has rank at most equal to $\Omega - 1$ and so there are at most $\Omega - 1$ nonzero eigenvalues. Therefore, we are unable to find more than $\Omega - 1$ discriminant functions (see, e.g., Bishop 2006).

Cross-References

▶ Regression
▶ Support Vector Machines

Recommended Reading

Most good statistical text books cover this.

Bellman RE (1961) Adaptive control processes. Princeton University Press, Princeton

Bishop C (2006) Pattern recognition and machine learning. Springer, New York

Duda RO, Hart PE (1973) Pattern classification and scene analysis. Wiley, New York

Fukunaga K (1990) Introduction to statistical pattern recognition, 2nd edn. Academic, San Diego

Linear Regression

Novi Quadrianto[1] and Wray L. Buntine[2,3]
[1]Department of Informatics, SMiLe CLiNiC, University of Sussex, Brighton, UK
[2]Statistical Machine Learning Program, NICTA, Canberra, ACT, Australia
[3]Faculty of Information Technology, Monash University, Clayton, VIC, Australia

Definition

Linear regression is an instance of the ▶ Regression problem which is an approach to modeling a functional relationship between input variables x and an output/response variable y. In linear regression, a linear function of the input variables is used, and more generally a linear function of some vector function of the input variables $\boldsymbol{\phi}(x)$ can also be used. The linear function estimates the mean of y (or more generally the median or a quantile).

Motivation and Background

Assume we are given a set of data points sampled from an underlying but unknown distribution, each of which includes input x and output

y. The task of regression is to learn a hidden functional relationship between x and y from observed and possibly noisy data points, so y is to be approximated in some way by $f(x)$. This is the task covered in more detail in Regression. A general approach to the problem is to make the function $f()$ be linear. Depending on the optimization criteria used to fit between the linear function $f(x)$ and the output y, this can include many different regression techniques, but optimization is generally easier because of the linearity.

Theory/Solution

Formally, in a regression problem, we are interested in recovering a functional dependency $y_i = f(x_i) + \epsilon_i$ from N observed training data points $\{(x_i, y_i)\}_{i=1}^N$, where $y_i \in \mathbb{R}$ is the noisy observed output at input location $x_i \in \mathbb{R}^d$. For the linear parametric technique, we tackle this regression problem by parameterizing the latent regression function $f()$ by a parameter $w \in \mathbb{R}^H$, that is, $f(x_i) := \langle \phi(x_i), w \rangle$ for H fixed basis functions $\{\phi_h(x_i)\}_{h=1}^H$. Note that the function is a linear function of the weight vector w. The simplest form of the linear parametric model is when $\phi(x_i) = x_i \in \mathbb{R}^d$, that is, the model is also linear with respect to the input variables, $f(x_i) := w_0 + w_1 x_i^1 + \cdots + w_d x_i^d$. Here the weight w_0 allows for any constant offset in the data. With general basis functions such as polynomials, exponentials, sigmoids, or even more sophisticated Fourier or wavelets bases, we can obtain a regression function which is nonlinear with respect to the input variables although still linear with respect to the parameters.

In the subsequent section, the simplest and thus common linear parametric method for solving a regression problem is covered, the least squares method.

Least Squares Method

Let $X \in \mathbb{R}^{N \times d}$ be a matrix of input variables and $y \in \mathbb{R}^N$ be a vector of output variables. The least squares method minimizes the following sum of squared error:

$$E(w) = (Xw - y)^T (Xw - y) \tag{1}$$

to infer the weight vector w. Note that the above error function is quadratic in the w, thus the minimization has a unique solution and leads to a closed-form expression for the estimated value of the unknown weight vector w. The minimizer of the error function in (1) can be found by setting its first derivative with respect to the weight vector to zero, that is,

$$\partial_w E(w) = 2X^T (Xw - y) = 0 \tag{2}$$

$$w^* = (X^T X)^{-1} X^T y. \tag{3}$$

The term

$$(X^T X)^{-1} X^T := X^\dagger \tag{4}$$

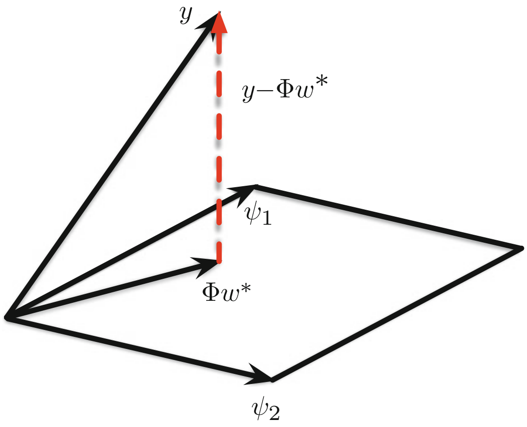
is known as the Moore-Penrose pseudo-inverse (Golub and Van Loan 1996) of the matrix X. This quantity can be regarded as a generalization of a matrix inverse to nonsquare matrices. Whenever X is square and invertible, $X^\dagger \equiv X^{-1}$. Having computed the optimal weight vector, we can then predict the output value at a novel input location x_{new} simply by taking an inner product: $y_{\text{new}} = \langle \phi(x_{\text{new}}), w^* \rangle$.

Under the assumption of an independent and normally distributed noise term, $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$, the above least squares approach can be shown to be equivalent to the maximum likelihood solution. With the Gaussian noise term, the log-likelihood model on an output vector y and an input matrix X is

$$\ln p(y|X, w) = \ln \mathcal{N}(Xw, \sigma^2 I) \tag{5}$$

$$= -\frac{N}{2} \ln(2\pi\sigma^2) - \frac{1}{2\sigma^2} (y - Xw)^T (y - Xw). \tag{6}$$

Maximizing the above likelihood function with respect to w will give the optimal weight to be in the form of (3). We can also find the maximum likelihood estimate of the noise variance by setting the first derivative of (6) with respect to σ^2 to zero, that is,



Linear Regression, Fig. 1 Geometrical interpretation of least squares (Bishop 2006). The optimal solution w^* with respect to the least squares criterion corresponds to the orthogonal projection of y onto the linear subspace which is formed by the vectors of the basis functions

$$\sigma^2_{\text{ML}} = \frac{1}{N}(y - Xw)^T(y - Xw). \tag{7}$$

Geometrical Interpretation of Least Squares Method

Let y be a vector in an N-dimensional space whose axes are given by $\{y_i\}_{i=1}^N$. Each of the H basis functions evaluated at N input locations can also be represented as a vector in the same N-dimensional space. For notational convenience, we denote this vector as ψ_h. The H vectors ψ_h will span a linear subspace of dimensionality H whenever the number of basis functions H is smaller than the number of input locations N (see Fig. 1). Denote $\Phi \in \mathbb{R}^{N\times H}$ as a matrix whose rows are the vectors $\{\phi_h(x_i)\}_{h=1}^H$. Our linear prediction model, Φw (in the simplest form Xw) will be an arbitrary linear combination of the vectors ψ_h. Thus, it can lie anywhere in the H-dimensional space. The sum of squared error criterion in (1) then corresponds to the squared Euclidean distance between Φw and y. Therefore, the least squares solution of w corresponds to the orthogonal projection of y onto the linear subspace. This orthogonal projection is associated with the minimum of the squared Euclidean distance. As a side note, from Fig. 1, it is clear that the vector $y - \Phi w$ is normal (perpendicular) to the range of Φ; thus $\Phi^T\Phi w = \Phi^T y$ is called the normal equation associated with the least squares problem.

Practical Note

The computation of (3) requires an inversion of an H by H matrix $\Phi^T\Phi$ (or a d by d matrix X^TX). A direct inversion of this matrix might lead to numerical difficulties when two or more basis vectors ψ_h or input dimensions are (nearly) collinear. This problem can be addressed conveniently by using singular value decomposition (SVD) (Press et al. 1992). It is important to note that adding a regularization term (see also the later section on ridge regression) ensures the nonsingularity of $\Phi^T\Phi$ matrix, even in the presence of degeneracies.

Sequential Learning of Least Squares Method

Computation of the optimal weight vector in (3) involves the whole training set comprising N data points. This learning technique is known as a batch algorithm. Real datasets can however involve large numbers of data points which might make batch techniques computationally prohibitive. In contrast, sequential algorithms or online algorithms process one data point at a time and can be more suited to handle large datasets.

We can use a sequential algorithm called stochastic gradient descent for learning the optimal weight vector. The objective function of (1) can be decomposed into $\sum_{i=1}^N(\langle x_i, w\rangle - y_i)^2$. This transformation suggests a simple stochastic gradient descent procedure: we traverse the data point i and update the weight vector using

$$w^{t+1} \leftarrow w^t - 2\eta(\langle x_i, w^t\rangle - y_i)x_i, \tag{8}$$

This algorithm is known as least mean squares (LMS) algorithm. In the above equation, t denotes the iteration number, and η denotes the learning rate. The value of η needs to be chosen carefully to ensure the convergence of the algorithm.

Regularized/Penalized Least Squares Method

The issue of over-fitting as mentioned in Regression is usually addressed by introducing a regularization or penalty term to the objective function. The regularized objective function is now in the form of

$$E_{\text{reg}} = E(w) + \lambda R(w). \tag{9}$$

Here $E(w)$ measures the quality (such as least squares quality) of the solution on the observed data points, $R(w)$ penalizes complex solutions, and λ is called the regularization parameter which controls the relative importance between the two. This regularized formulation is sometimes called *coefficient shrinkage* as it shrinks coefficients/weights toward zero (cf. *coefficient subset selection* formulation where the best k out of H basis functions are greedily selected). Two simple penalty terms $R(w)$ are given next, but more generally measures of curvature can also be used to penalize non-smooth functions.

Ridge Regression

The regularization term is in the form of

$$R(w) = \sum_{d=1}^{D} w_d^2. \tag{10}$$

Considering $E(w)$ to be in the form of (1), the regularized least squares quality function is now

$$(Xw - y)^T(Xw - y) + \lambda w^T w. \tag{11}$$

Since the additional term is a quadratic of w, the regularized objective function is still quadratic in w, thus the optimal solution is unique and can be found in closed form. As before, setting the first derivative of (11) with respect to w to zero, the optimal weight vector is in the form of

$$\partial_w E_{\text{reg}}(w) = 2X^T(Xw - y) + 2\lambda w = 0 \tag{12}$$

$$w^* = (X^T X + \lambda I)^{-1} X^T y. \tag{13}$$

The effect of the regularization term is to put a small weight for those basis functions which are useful only in a minor way as the penalty for small weights is very small.

Lasso Regression

The regularization term is in the form of

$$R(w) = \sum_{d=1}^{D} |w_d|. \tag{14}$$

In contrast to ridge regression, lasso regression (Tibshirani 1996) has no closed-form solution. In fact, the non-differentiability of the regularization term has produced many approaches. Most of the methods involve quadratic programming and recently coordinate-wise descent algorithms for large lasso problems (Friedman et al. 2007). Lasso regression leads to sparsity in w, that is, only a subset of w is nonzero, so irrelevant basis functions will be ignored.

Cross-References

▶ Gaussian Processes
▶ Regression

Recommended Reading

Statistical textbooks and machine learning textbooks, such as Bishop (2006) among others, introduce different linear regression models. For a large variety of built-in linear regression techniques, refer to R (http://www.r-project.org/).

Bishop C (2006) Pattern recognition and machine learning. Springer, New York

Friedman J, Hastie T, Hölfling H, Tibshirani R (2007) Pathwise coordinate optimization. Ann Stat 1(2):302–332

Golub GH, Van Loan CF (1996) Matrix computations, 3rd edn. John Hopkins University Press, Baltimore

Press WH, Teukolsky SA, Vetterling WT, Flannery BP (1992) Numerical recipes in C: the art of scientific computing, 2nd edn. Cambridge University Press, Cambridge. ISBN:0-521-43108-5

Tibshirani R (1996) Regression shrinkage and selection via the lasso. J R Stat Soc Ser B Stat Methodol 58:267–288

Linear Regression Trees

► Model Trees

Linear Separability

Two classes are linearly separable if there exists a hyperplane that separates the data for each of the classes.

Cross-References

► Precision
► Support Vector Machines

Link Analysis

► Link Mining and Link Discovery

Link Mining and Link Discovery

Lise Getoor
University of Maryland, College Park, MD, USA

Synonyms

Link analysis; Network analysis

Definition

Many domains of interest today are best described as a linked collection of interrelated objects. Datasets describing these domains may describe homogeneous networks, in which there is a single-object type and link type, or richer, heterogeneous networks, in which there may be multiple object and link types (and possibly other semantic information). Examples of homogeneous networks include social networks, such as people connected by friendship links, or the WWW, a collection of linked web pages. Examples of heterogeneous networks include those in medical domains describing patients, diseases, treatments and contacts, or bibliographic domains describing publications, authors, and venues. *Link mining* refers to data mining techniques that explicitly consider these links when building predictive or descriptive models of the linked data. Commonly addressed link mining tasks include collective classification, object ranking, group detection, link prediction, and subgraph discovery. Additional important components include entity resolution, and other data cleaning and data mapping operations.

Motivation and Background

"Links," or more generically "relationships," among data instances are ubiquitous. These links often exhibit patterns that can indicate properties of the data instances such as the importance, rank, or category of the instances. In some cases, not all links will be observed; therefore, we may be interested in predicting the existence of links between instances. Or, we may be interested in identifying unusual or anomalous links. In other domains, where the links are evolving over time, our goal may be to predict whether a link will exist in the future, given the previously observed links. By taking links into account, more complex patterns may be discernable as well. This observation leads to other challenges focused on discovering substructures, such as communities, groups, or common subgraphs. In addition, links can also help in the process of ► entity resolution, or figuring out when two instance references refer to the same underlying entity.

Link mining is a newly emerging research area that is at the intersection of the work in link analysis (Feldman 2002; Jensen and Goldberg 1998) hypertext and web mining (Chakrabarti 2002), ► relational learning and ► inductive logic programming (Raedt 2008), and ► graph mining (Cook and Holder 2000). We use the term link mining to put a special emphasis on the links – moving them up to first-class citizens in the data analysis endeavor.

Theory/Solution

Traditional data mining algorithms such as ▶ association rule mining, market basket analysis, and cluster analysis commonly attempt to find patterns in a dataset characterized by a collection of independent instances of a single relation. This is consistent with the classical statistical inference problem of trying to identify a model given an independent, identically distributed (IID) sample. One can think of this process as learning a model for the node attributes of a homogeneous graph while ignoring the links between the nodes.

A key emerging challenge for data mining is tackling the problem of mining richly structured, heterogeneous datasets. These kinds of datasets are commonly described as networks or graphs. The domains often consist of a variety of object types; the objects can be linked in a variety of ways. Thus, the graph may have different node and edge (or hyperedge) types. Naively applying traditional statistical inference procedures, which assume that instances are independent, can lead to inappropriate conclusions about the data (Jensen 1999). Care must be taken that potential correlations due to links are handled appropriately. In fact, object linkage is knowledge that should be exploited. This information can be used to improve the predictive accuracy of the learned models: attributes of linked objects are often correlated, and links are more likely to exist between objects that have some commonality. In addition, the graph structure itself may be an important element to include in the model. Structural properties such as degree and connectivity can be important indicators.

Data Representation

While data representation and feature selection are significant issues for traditional machine learning algorithms, data representation for linked data is even more complex. Consider a simple example from Singh et al. (2005) of a social network describing actors and their participation in events. Such social networks are commonly called *affiliation networks* (Wasserman and Faust 1994), and are easily represented by three tables representing the actors, the events, and the participation relationships. Even this simple structure can be represented as several distinct graphs. The most natural representation is a bipartite graph, with a set of actor nodes, a set of event nodes, and edges that represent an actor's participation in an event. Other representations may enable different insights and analysis. For example, we may construct a network in which the actors are nodes and edges correspond to actors who have participated in an event together. This representation allows us to perform a more actor-centric analysis. Alternatively, we may represent these relations as a graph in which the events are nodes, and events are linked if they have an actor in common. This representation may allow us to more easily see connections between events.

This flexibility in the representation of a graph arises from a basic graph representation duality. This duality is illustrated by the following simple example: Consider a data set represented as a simple $G = (\mathbf{0}, \mathbf{L})$, where $\mathbf{0}$ is the set of objects (i.e., the nodes or vertices) and $\mathbf{L}$ is the set of links (i.e., the edges or hyperedges). The graph $G(\mathbf{0}, \mathbf{L})$ can be transformed into a new graph $G'(\mathbf{0}', \mathbf{L}')$, in which the links l_i, l_j in G are objects in G' and there exists an link between $o_i, o_j \in \mathbf{0}'$ if and only if l_i and l_j share an object in G. This basic graph duality illustrates one kind of simple data representation transformation. For graphs with multiple node and edge types, the number of possible transformations becomes immense. Typically, these reformulations are not considered as part of the link mining process. However, the representation chosen can have a significant impact on the quality of the statistical inferences that can be made. Therefore, the choice of an appropriate representation is actually an important issue in effective link mining, and is often more complex than in the case where we have IID data instances.

Link Mining and Link Discovery, Table 1 A simple categorization of different link mining tasks

1. Object-related tasks
a. Object classification (collective classification)
b. Object clustering (group detection)
c. Object consolidation (entity resolution)
d. Object ranking
2. Link-related tasks
a. Link labeling/classification
b. Link prediction
c. Link ranking
3. Graph-related tasks
a. Subgraph discovery
b. Graph classification

Link Mining Tasks

Link mining puts a new twist on some classic data mining tasks, and also poses new problems. One way to understand the different types of learning and inference problems is to categorize them in terms of the components of the data that are being targeted. Table 1 gives a simple characterization. Note that for the object-related tasks, even though we are concerned with classifying, clustering, consolidating, or ranking the objects, we will be exploiting the links. Similarly for link-related tasks, we can use information about the objects that participate in the links, and their links to other objects and so on.

In addition, because of the underlying link structure, link mining affords the opportunity for inferences and predictions to be *collective* or dependent on one another. The simplest example of this is in collective classification, where the inferred label of one node can depend on the inferred label of its neighbors. There are a variety of ways of modeling and exploiting this dependence. Methods include performing joint inference in the appropriate probabilistic model, use of information diffusion models, constructing and optimizing the appropriate structured prediction using a max margin approach, and others.

Additional information on different link mining subtasks is provided in separate entries on *collective classification*, *entity resolution*, *group detection*, and *link prediction*. Related problems and techniques can be found in the entries on *relational learning*, *graph mining*, and *inductive logic programming*.

Cross-References

- ▸ Collective Classification
- ▸ Entity Resolution
- ▸ Graph Clustering
- ▸ Graph Mining
- ▸ Group Detection
- ▸ Inductive Logic Programming
- ▸ Link Prediction
- ▸ Relational Learning

Recommended Reading

Chakrabarti S (2002) Mining the web. Morgan Kaufman, San Francisco

Cook DJ, Holder LB (2000) Graph-based data mining. IEEE Intell Syst 15(2):32–41. ISSN:1094–7167. http://dx.doi.org/10.1109/5254.850825

Feldman R (2002) Link analysis: current state of the art. In: Proceedings of the KDD'02, Edmonton

Jensen D (1999) Statistical challenges to inductive inference in linked data. In: Seventh international workshop on artificial intelligence and statistics, Fort Lauderdale. Morgan Kaufman, San Francisco

Jensen D, Goldberg H (1998) AAAI fall symposium on AI and link analysis, Orlando. AAAI Press, Menlo Park

Raedt LD (ed) (2008) Logical and relational learning. Springer, Berlin

Singh L, Getoor L, Licamele L (2005) Pruning social networks using structural properties and descriptive attributes. In: International conference on data mining, 2005, Houston. IEEE Computer Society

Wasserman S, Faust K (1994) Social network analysis: methods and applications. Cambridge University Press, Cambridge

Link Prediction

Galileo Namata and Lise Getoor
University of Maryland, College Park, MD, USA

Synonyms

Edge prediction; Relationship extraction

Definition

Many datasets can naturally be represented as graph where nodes represent instances and links represent relationships between those instances. A fundamental problem with these types of data is that the link information in the graph may be of dubious quality; links may incorrectly exist between unrelated nodes and links may be missing between two related nodes. The goal of link prediction is to predict the existence of incorrect or missing links between the nodes of the graph.

Theory/Solution

Inferring the existences of edges between nodes in a graph has traditionally been referred to as *link prediction* (Liben-Nowell and Kleinberg 2003a; Taskar et al. 2003). Link prediction is a challenging problem that has been studied in various guises in different domains. For example, in social network analysis, there is work on predicting friendship links (Zheleva et al. 2008), event participation links (i.e., coauthorship O'Madadhain et al. 2005), communication links (i.e., email O'Madadhain et al. 2005), and links representing semantic relationships (i.e., advisor of Taskar et al. 2003, subordinate manager Diehl et al. 2007). In bioinformatics, there is interest in predicting the existence of edges representing physical protein–protein interactions (Szilagyi et al. 2005b; Yu et al. 2006), domain–domain interactions (Deng et al. 2002), and regulatory interactions (Albert et al. 2007). Similarly, in computer network systems, there is work in inferring unobserved connections between routers, as well as inferring relationships between autonomous systems and service providers (Spring et al. 2004). There is also work on using link prediction to improve recommender systems (Farrell et al. 2005), Web site navigation (Zhu 2003), surveillance (Huang and Lin 2008), and automatic document cross-referencing (Milne and Witten 2008).

We begin with some basic definitions and notation. We refer to the set of possible edges in a graph as *potential edges*. The set of potential edges depends on the graph type and how the edges for the graph are defined. For example, in a directed graph, the set of potential edges consists of all edges $e = (v_1, v_2)$ where v_1 and v_2 are any two nodes V in the graph (i.e., the number of potential edges is $|V| \times |V|$). In an undirected bipartite graph with two subsets of nodes $(V_1, V_2 \in V)$, while the edges still consist of a pair of nodes, $e = (v_1, v_2)$, there is an added condition such that one node must be from V_1 and the other node must be from V_2; this results in $|V_1| \times |V_2|$ potential edges. Next, we refer to set of "true" edges in a graph as *positive edges*, and we refer to the "true" non-edges in a graph (i.e., pairs of nodes without edges between them) as *negative edges*. For a given graph, typically we only have information about a subset of the edges; we refer to this set as the *observed* edges. The observed edges can include both positive and negative edges, though in many formulations there is an assumption of positive-only information. We can view link prediction as a probabilistic inference problem, where the evidence includes the observed edges, the attribute values of the nodes involved in the potential edge, and possibly other information about the network, and for any unobserved, potential edge, we want to compute the probability of it existing. This can be reframed as a binary classification problem by choosing some probability threshold and concluding that potential edges with existence probability above the threshold are true edges and those below the threshold are considered false edges (more complex schemes are possible as well). For noisy and incomplete networks, we use terminology from the machine learning literature and refer to an edge that is inferred to exist and is a true edge in the graph as a *true positive edge*, an edge that should exist but is not inferred as a *false negative edge*, an edge that should not exist and is not inferred as a *true negative edge*, and an edge that should not exist but is incorrectly inferred to exist as a *false positive edge*.

One of the early and simple formulations of the link prediction problem was proposed by Liben-Nowell and Kleinberg (2003b). They proposed a temporal prediction problem defined over a dynamic network where given a graph

$G_t(V_t, E_t)$ at time t, the problem is to infer the set of edges at the next time step $t + 1$. More formally, the objective is to infer a set of edges E_{new} where $E_{t+1} = E_t \bigcup E_{\text{new}}$. We use a more general definition of link prediction proposed by Taskar et al. (2003) where given a graph G and the set of potential edges in G, denoted $P(G)$, the problem of link prediction is to predict for all $p \in P(G)$ whether p exists or does not exist, remaining agnostic on whether G is a noisy graph with missing edges or a snapshot of a dynamic graph at a particular time point.

Approaches

In this section, we discuss the two general categories of the current link prediction models: topology-based approaches and node attribute-based approaches. Topology-based approaches are methods that rely solely on the topology of the network to infer edges. Node attribute-based approaches make predictions based on the attribute values of the nodes incident to the edges. In addition, there are models that make use of both structure and attribute values.

Topology-Based Approaches

A number of link prediction models have been proposed, which rely solely on the topology of the network. These models typically rely on some notion of structural proximity, where nodes that are close are likely to share an edge (e.g., sharing common neighbors, nodes with a small shortest path distance between). The earliest topological approach for link prediction was proposed by Liben-Nowell and Kleinberg (2003b). In this work, Liben-Nowell and Kleinberg proposed various structure-based similarity scores and applied them over the unobserved edges of an undirected graph. They then use a threshold k and only predict edges with the top k scores as existing. A variety of similarity scores were proposed, given two nodes v_1 and v_2, including graph distance (the length of the shortest path between v_1 and v_2), common neighbors (the size of the intersection of the sets of neighbors of v_1 and v_2), and more complex measures such as the Katz measure (the sum of the lengths of the paths between v_1 and v_2 exponentially damped by length to count short paths more heavily). Evaluating over a coauthorship network, the best performing proximity score measure was the Katz measure; however the simple measures, which rely only on the intersection of the set of nodes adjacent to both nodes, performed surprisingly well. A related approach was proposed by Yu et al. (2006), which applies the link prediction problem to predicting missing protein–protein interactions (PPI) from PPI networks generated by high-throughput methods. This work assumes that interacting proteins tend to form a clique. Thus, missing edges can be predicted by predicting the existence of edges that will create cliques in the network. More recent work by Clauset et al. (2008) has tried to go beyond predicting edges between neighboring nodes. In their problem domain of food webs, for example, pairs of predators often prey on a shared prey species but rarely prey on each other. Thus, in these networks, predicting "predator–prey" edges need to go beyond proximity. For this, they propose a "hierarchical random graph" approach, which fits a hierarchical model to all possible dendrograms of a given network. The model is then used to calculate the likelihood of an edge existing in the network.

Node Attribute-Based Approaches

Although topology is useful in link prediction, topology-based approaches ignore an important source of information in networks, the attributes of nodes. Often there are correlations in the attributes of nodes that share an edge with each other. One approach that exploits this correlation was proposed by Taskar et al. (2003). In their approach, Taskar et al. (2003) applied the relational Markov network (RMN) framework to link prediction to predicting the existence and class of edges between Web sites. They exploit the fact that certain links can only exist between nodes of the appropriate type. For example, an "advisor" edge can only exist between student and faculty.

Another approach that uses node attributes was proposed by Popescul and Ungar (2003). In that approach, they used a structured ▶ logistic regression model over learned relational features to predict citation edges in a citation network. Their relational features are built over attributes such as the words used in the paper nodes. O'Madadhain et al. (2005) also approached an attribute-based approach, constructing local conditional probability models based on the attributes such as node attribute similarity, topic distribution, and geographical location in predicting "co-participation" edges in an email communication network. More recently, there is work on exploiting other node attributes like the group membership of the nodes. Zheleva et al. (2008) showed that membership in family groups is very useful in predicting friendship links in social networks. Similarly, Sprinzak et al. (2006) showed that using protein complex information can be useful in predicting protein–protein interactions. Finally, we note that in link prediction, as in classification, the quality of predictions can be improved by making the predictions collectively. Aside from the relational Markov network approach by Taskar et al. (2003) mentioned earlier, Markov logic networks (Richardson and Domingos 2006) and probabilistic relational models (Getoor et al. 2003) have also been proposed for link prediction and are capable of performing joint inference.

Issues

There are a number of challenges that make link prediction very difficult. The most difficult challenge is the large class skew between the number of edges that exist and the number of edges that do not. To illustrate, consider directed graph denoted by $G(V, E)$. While the number of edges $|E|$ is often $O(|V|)$, the number of edges that do not exist is often $O(|V|^2)$. Consequently, the prior probability edge existence is very small. This causes many supervised models, which naively optimize for accuracy, to learn a trivial model, which always predicts that a link does not exist. A related problem in link prediction is the large number of edges whose existence must be considered. The number of potential edges is $O(|V|^2)$ and this limits the size of the datasets that can be considered.

In practice, there are general approaches to addressing these issues either prior to or during the link prediction. With both large class skew and number of edges to contend with, the general approach is to make assumptions that reduce the number of edges to consider. One common way to do this is to partition the set of nodes where we only consider potential edges between nodes of the same partition; edges between partitions are not explicitly modeled, but are assumed not to exist. This is useful in many domains where there is some sort of natural partition among the nodes available (e.g., geography in social networks, location of proteins in a cell), which make edges across partitions unlikely. Another way is to define some simple, computationally inexpensive distance measure such that only edges whose nodes are within some distance are considered.

Another practical issue in link prediction is that while real-world data often indicates which edges exist (positive examples), the edges which do not exist (negative examples) are rarely annotated for use by link prediction models. In bioinformatics, for example, the protein–protein interaction network of yeast, the most and annotated studied organism, is annotated with thousands of observed edges (physical interactions) between the nodes (proteins) gathered from numerous experiments. There are currently, however, no major datasets available that indicate which proteins definitely do not physically interact. This is an issue not only in creating and learning models for link prediction but is also an issue evaluating them. Often, it is unclear whether a predicted edge which is not in our ground truth data is an incorrectly predicted edge or an edge resulting from incomplete data.

Related Problems

In addition to the definition of link prediction discussed above, it is also important to mention three closely related problems: *link completion*, *leak detection*, and *anomalous link discovery*, whose objectives are different but very similar to link prediction. Link completion (Chaiwanarom and Lursinsap 2008; Goldenberg et al. 2003) and leak detection (Balasubramanyan et al. 2009; Carvalho and Cohen 2007) are a variation of link prediction over hypergraphs. A hypergraph is a graph where the edges (known as hyperedges) can connect any number of nodes. For example, in a hypergraph representing an email communication network, a hyperedge may connect nodes representing email addresses that are recipients of a particular email communication. In link completion, given the set of nodes that participate in a particular hyperedge, the objective is to infer nodes that are missing. For the email communication network example, link completion may involve inferring which email addresses need to be added to the recipient list of an email communication. Conversely, in leak detection, given the set of nodes participating in a particular hyperedge, the objective is to infer which nodes should not be part of that hyperedge. For example, in email communications, leak detection will attempt to infer which email address nodes are incorrectly part of the hyperedge representing the recipient list of the email communication.

The last problem, anomalous link discovery (Huang and Zeng 2006; Rattigan and Jensen 2005a), has been proposed as an alternate task to link prediction. As with link completion, the existence of the edges is assumed to be observed, and the objective is to infer which of the observed links are anomalous or unusual. Specifically, anomalous link discovery identifies which links are statistically improbable with the idea that these may be of interest for those analyzing the network. Rattigan and Jensen (2005b) show that some methods that perform poorly for link prediction can still perform well for anomalous link discovery.

Cross-References

▶ Graph Mining
▶ Statistical Relational Learning

Recommended Reading

Albert R, DasGupta B, Dondi R, Kachalo S, Sontag E, Zelikovsky A et al (2007) A novel method for signal transduction network inference from indirect experimental evidence. J Comput Biol 14:407–419

Balasubramanyan R, Carvalho VR, Cohen W (2009) Cutonce recipient recommendation and leak detection in action. In: Workshop on enhanced messaging, Chicago

Carvalho VR, Cohen WW (2007) Preventing information leaks in email. In: SIAM conference on data mining, Minneapolis

Chaiwanarom P, Lursinsap C (2008) Link completion using prediction by partial matching. In: International symposium on communications and information technologies, Vientiane

Clauset A, Moore C, Newman MEJ (2008) Hierarchical structure and the prediction of missing links in networks. Nature 453:98

Deng M, Mehta S, Sun F, Chen T (2002) Inferring domain-domain interactions from protein-protein interactions. Genome Res 12(10):1540–1548

Diehl C, Namata GM, Getoor L (2007) Relationship identification for social network discovery. In: Proceedings of the 22nd national conference on artificial intelligence, Vancouver

Farrell S, Campbell C, Myagmar S (2005) Relescope: an experiment in accelerating relationships. In: Extended abstracts on human factors in computing systems, Portland

Getoor L, Friedman N, Koller D, Taskar B (2003) Learning probabilistic models of link structure. Mach Learn 3:679–707

Goldenberg A, Kubica J, Komarek P, Moore A, Schneider J (2003) A comparison of statistical and machine learning algorithms on the task of link completion. In: Conference on knowledge discovery and data mining, workshop on link analysis for detecting complex behavior, Washington, DC

Huang Z, Lin DKJ (2008) The time-series link prediction problem with applications in communication surveillance. Inf J Comput 21:286–303

Huang Z, Zeng DD (2006) A link prediction approach to anomalous email detection. In: IEEE international conference on systems, man, and cybernetics, Taipei

Liben-Nowell D, Kleinberg J (2003a) The link prediction problem for social networks. In: International conference on information and knowledge management, New Orleans

Liben-Nowell and Kleinberg (2003b)

Milne D, Witten IH (2008) Learning to link with wikipedia. In: Proceedings of the 17th ACM conference on information and knowledge management, Napa Valley

O'Madadhain J, Hutchins J, Smyth P (2005) Prediction and ranking algorithms for event-based network data. SIGKDD Explor Newsl 7(2):23–30

Popescul A, Ungar LH (2003) Statistical relational learning for link prediction. In: International joint conferences on artificial intelligence workshop on learning statistical models from relational data
Rattigan MJ, Jensen D (2005a) The case for anomalous link discovery. SIGKDD Explor Newsl 7:41–47
Rattigan and Jensen (2005b)
Richardson M, Domingos P (2006) Markov logic networks. Mach Learn 62:107–136
Spring N, Wetherall D, Anderson T (2004) Reverse engineering the internet. SIGCOMM Comput Commun Rev 34(1):3–8
Sprinzak E, Altuvia Y, Margalit H (2006) Characterization and prediction of protein-protein interactions within and between complexes. Proc Natl Acad Sci 103(40):14718–14723
Szilagyi A, Grimm V, Arakaki AK, Skolnick J (2005a) Prediction of physical protein-protein interactions. Phys Biol 2(2):S1–S16
Szilagyi et al. (2005b)
Taskar B, Wong M-F, Abbeel P, Koller D (2003) Link prediction in relational data. In: Advances in neural information processing systems, Vancouver
Yu H, Paccanaro A, Trifonov V, Gerstein M (2006) Predicting interactions in protein networks by completing defective cliques. Bioinformatics 22(7):823–829
Zheleva E, Getoor L, Golbeck J, Kuter U (2008) Using friendship ties and family circles for link prediction. In: 2nd ACM SIGKDD workshop on social network mining and analysis, Las Vegas
Zhu J (2003) Mining web site link structure for adaptive web site navigation and search. Ph.D. thesis, University of Ulster at Jordanstown

Link-Based Classification

▸ Collective Classification

Liquid State Machine

▸ Reservoir Computing

List Washing

▸ Record Linkage

Local Distance Metric Adaptation

Synonyms

Kernel shaping; Nonstationary kernels; Supersmoothing

Definition

In learning systems with kernels, the shape and size of a kernel plays a critical role for accuracy and generalization. Most kernels have a distance metric parameter, which determines the size and shape of the kernel in the sense of a Mahalanobis distance. Advanced kernel learning tune every kernel's distance metric individually, instead of turning one global distance metric for all kernels.

Cross-References

▸ Locally Weighted Regression for Control

Local Feature Selection

▸ Projective Clustering

Locality Sensitive Hashing Based Clustering

Xin Jin[1] and Jiawei Han[2]
[1]PayPal Inc., San Jose, CA, USA
[2]University of Illinois at Urbana-Champaign, Urbana, IL, USA

The basic idea of the LSH (Gionis et al. 1999) technique is using multiple hash functions to hash the data points and guarantee that there is a high probability of collision for points which are close to each other and low collision probability for dissimilar points. LSH schemes exist for many distance measures, such as Hamming norm, L_p norms, cosine distance, earth movers distance (EMD), and Jaccard coefficient.

In LSH, define a family $H = \{h : S \to U\}$ as locality-sensitive, if for any a, the function $p(t) = Pr_H[h(a) = h(b) : ||a - b|| = x]$ is decreasing in x. Based on this definition, the probability of collision of points a and b is decreasing with their distance.

Although LSH was originally proposed for approximate nearest neighbor search in high dimensions, it can be used for clustering as well (Das et al. 2007; Haveliwala et al. 2000). The buckets could be used as the bases for clustering. Seeding the hash functions several times can help getting better quality clustering.

Recommended Reading

Das AS, Datar M, Garg A, Rajaram S (2007) Google news personalization: scalable online collaborative filtering. In: Proceedings of the 16th international conference on world wide web (WWW'07). ACM, New York, pp 271–280

Gionis A, Indyk P, Motwani R (1999) Similarity search in high dimensions via hashing. In: Proceedings of the 25th international conference on very large data bases (VLDB'99). Morgan Kaufmann Publishers, San Francisco, pp 518–529

Haveliwala TH, Gionis A, Indyk P (2000) Scalable techniques for clustering the web (extended abstract). In: Proceedings of the third international workshop on the web and databases. Stanford University, Stanford, pp 129–134

Locally Weighted Learning

▶ Locally Weighted Regression for Control

Locally Weighted Regression for Control

Jo-Anne Ting[1], Franziska Meier[2], Sethu Vijayakumar[1,2], and Stefan Schaal[3,4]

[1]University of Edinburgh, Edinburgh, UK

[2]University of Southern California, Los Angeles, CA, USA

[3]Max Planck Institute for Intelligent Systems, Stuttgart, Germany

[4]Computer Science, University of Southern California, Los Angeles, CA, USA

Synonyms

Kernel shaping; Lazy learning; Locally weighted learning; Local distance metric adaptation; LWR; LWPR; Nonstationary kernels; Supersmoothing

Definition

This entry addresses two topics: ▶ learning control and locally weighted regression.

▶ Learning control refers to the process of acquiring a control strategy for a particular control system and a particular task by trial and error. It is usually distinguished from adaptive control (Aström and Wittenmark 1989) in that the learning system is permitted to fail during the process of learning, resembling how humans and animals acquire new movement strategies. In contrast, adaptive control emphasizes single-trial convergence without failure, fulfilling stringent performance constraints, e.g., as needed in life-critical systems like airplanes and industrial robots.

Locally weighted regression refers to ▶ supervised learning of continuous functions (otherwise known as function approximation or ▶ regression) by means of spatially localized algorithms, which are often discussed in the context of ▶ kernel regression, ▶ nearest neighbor methods, or ▶ lazy learning (Atkeson et al. 1997). Most regression algorithms are global learning systems. For instance, many algorithms can be understood in terms of minimizing a global ▶ loss function such as the expected sum squared error:

$$J = E\left[\frac{1}{2}\sum_{i=1}^{N}(\mathbf{t}_i - \mathbf{y}_i)^2\right] = E\left[\frac{1}{2}\sum_{i=1}^{N}\left(\mathbf{t}_i - \phi(\mathbf{x}_i)^T\boldsymbol{\beta}\right)^2\right] \qquad (1)$$

where $E[\cdot]$ denotes the expectation operator, $\mathbf{t}_i$ the noise-corrupted target value for an input $\mathbf{x}_i$—which is expanded by basis functions into a basis function vector $\phi(\mathbf{x}_i)$-and $\boldsymbol{\beta}$ is the vector of (usually linear) regression coefficients. Classical feedforward ▶ neural networks, ▶ radial basis function networks, ▶ mixture models, or ▶ Gaussian process regression are all global function approximators in the spirit of Eq. (1).

In contrast, local learning systems conceptually split up the global learning problem into multiple simpler learning problems. Traditional locally weighted regression approaches achieve this by dividing up the cost function into multiple independent local cost functions,

$$\begin{aligned} J &= E\left[\frac{1}{2}\sum_{k=1}^{K}\sum_{i=1}^{N} w_{k,i}\left(\mathbf{t}_i - \mathbf{x}_i^T\boldsymbol{\beta}_k\right)^2\right] \\ &= \frac{1}{2}\sum_{k=1}^{K} E\left[\sum_{i=1}^{N} w_{k,i}\left(\mathbf{t}_i - \mathbf{x}_i^T\boldsymbol{\beta}_k\right)^2\right] \\ &= \frac{1}{2}\sum_{k=1}^{K} J_k. \end{aligned} \qquad (2)$$

resulting in K (independent) local model learning problems. A different strategy for local learning starts out with the global objective (Eq. 1) and reformulates it to capture the idea of local models that cooperate to generate a (global) function fit. This is achieved by assuming there are K feature functions ϕ_k, such that the kth feature function $\phi_k(\mathbf{x}_i) = w_{k,i}\mathbf{x}_i$, resulting in

$$\begin{aligned} J &= E\left[\frac{1}{2}\sum_{i=1}^{N}\left(\mathbf{t}_i - \phi(\mathbf{x}_i)^T\boldsymbol{\beta}\right)^2\right] \\ &= E\left[\frac{1}{2}\sum_{i=1}^{N}\left(\mathbf{t}_i - \sum_{k=1}^{K} w_{k,i}(\mathbf{x}_i^T\boldsymbol{\beta}_k)\right)^2\right]. \end{aligned} \qquad (3)$$

In this setting, local models are initially coupled and approximations are found to decouple the learning of the local models parameters.

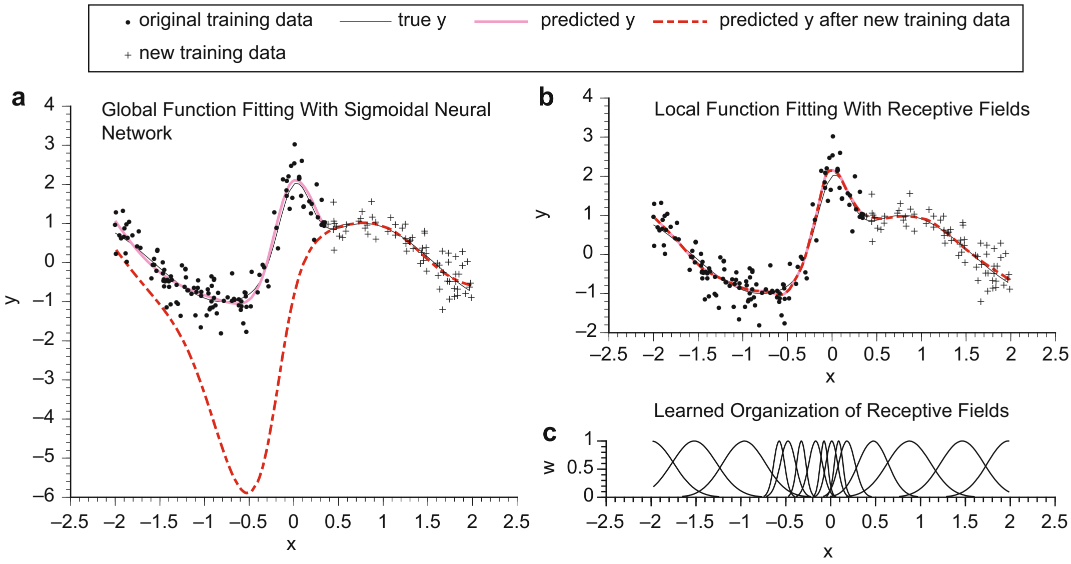
Motivation and Background

Figure 1 illustrates why locally weighted regression methods are often favored over global methods when it comes to learning from incrementally arriving data, especially when dealing with nonstationary input distributions. The figure shows the division of the training data into two sets: the "original training data" and the "new training data" (in dots and crosses, respectively).

Initially, a sigmoidal ▶ neural network and a locally weighted regression algorithm are trained on the "original training data," using 20 % of the data as a cross validation set to assess convergence of the learning. In a second phase, both learning systems are trained solely on the "new training data" (again with a similar cross-validation procedure), but without using any data from the "original training data." While both algorithms generalize well on the "new training data," the global learner incurred catastrophic interference, unlearning what was learned initially, as seen in Fig. 1a. Figure 1b shows that the locally weighted regression algorithm does not have this problem since learning (along with ▶ generalization) is restricted to a local area.

Appealing properties of locally weighted regression include the following:

- Function approximation can be performed incrementally with nonstationary input and output distributions and without significant danger of interference. Locally weighted regression can provide ▶ posterior probability



Locally Weighted Regression for Control, Fig. 1 Function approximation results for the function $y = \sin(2x) + 2\exp(-16x^2) + N(0, 0.16)$ with (**a**) a sigmoidal neural network, (**b**) a locally weighted regression algorithm (note that the data traces "true y," "predicted y," and "predicted y after new training data" largely coincide), and (**c**) the organization of the (Gaussian) kernels of (**b**) after training. See Schaal and Atkeson 1998 for more details

distributions, offer confidence assessments, and deal with heteroscedastic data.

- Locally weighted learning algorithms are computationally inexpensive to compute. It is well suited for online computations (e.g., for ▸ online and ▸ incremental learning) in the fast control loop of a robot—typically on the order of 100–1000 Hz.
- Locally weighted regression methods can implement continual learning and learning from large amounts of data without running into severe computational problems on modern computing hardware.
- Locally weighted regression is a nonparametric method (i.e., it does not require that the user determine a priori the number of local models in the learning system), and the learning systems grow with the complexity of the data it tries to model.
- Locally weighted regression can include ▸ feature selection, ▸ dimensionality reduction, and ▸ Bayesian inference—all which are required for robust ▸ statistical inference.
- Locally weighted regression works favorably with locally linear models (Hastie and Loader 1993), and local linearizations are of ubiquitous use in control applications.

Background

Returning to Eqs. (1) to (3), the main differences between global methods that directly solve Eq. (1) and local methods that solve either Eqs. (2) or (3) are listed below:

(i) A weight $w_{i,k}$ is introduced that focuses:
   - either the function approximation fit in Eq. (2)
   - or a local models contribution toward the global function fit in Eq. (3)

   on only a small neighborhood around a point of interest $\mathbf{c}_k$ in input space (see Eq. 4 below).

(ii) The learning problem is split into K independent optimization problems.

(iii) Due to the restricted scope of the function approximation problem, we do not need a nonlinear basis function expansion and can, instead, work with simple local functions or local polynomials (Hastie and Loader 1993).

The weights $w_{k,i}$ in Eq. (2) are typically computed from some ▶ kernel function (Atkeson et al. 1997) such as a squared exponential kernel:

$$w_{k,i} = \exp\left(-\frac{1}{2}\left(\mathbf{x}_i - \mathbf{c}_k\right)^T \mathbf{D}_k \left(\mathbf{x}_i - \mathbf{c}_k\right)\right) \quad (4)$$

with $\mathbf{D}_k$ denoting a positive semidefinite distance metric and $\mathbf{c}_k$ the center of the kernel. The number of kernels K is not finite. In many local learning algorithms, the kernels are never maintained in memory. Instead, for every query point $\mathbf{x}_q$, a new kernel is centered at $\mathbf{c}_k = \mathbf{x}_q$, and the localized function approximation is solved with weighted ▶ regression techniques (Atkeson et al. 1997).

Locally weighted regression should not be confused with mixture of experts models (Jordan and Jacobs 1994). ▶ Mixture models are *global* learning systems since the experts compete globally to cover training data. Mixture models address the ▶ bias-variance dilemma (Intuitively, the ▶ bias-variance dilemma addresses how many parameters to use for a function approximation problem to find an optimal balance between ▶ overfitting and oversmoothing of the training data.) by finding the right number of local experts. Locally weighted regression addresses the ▶ bias-variance dilemma in a local way by finding the optimal distance metric for computing the weights in the locally weighted regression (Schaal and Atkeson 1998).

Structure of Learning System

All local learning approaches have three critical components in common:

(i) Optimizing the regression parameters $\boldsymbol{\beta}_k$
(ii) Learning the distance metric $\mathbf{D}_k$ that defines a local model neighborhood
(iii) Choosing the location $\mathbf{c}_k$ of receptive field(s)

Local learning methods can be separated into "lazy" approaches that require all training data to be stored and "memoryless" approaches that compress data into a several local models and thus do not require storage of data points.

In the "lazy" approach, the computational burden of a prediction is deferred until the last moment, i.e., when a prediction is needed. Such a "compute-the-prediction-on-the-fly" approach is often called lazy learning and is a memory-based learning system where all training data is kept in memory for making predictions. A prediction is formed by optimizing the parameters $\boldsymbol{\beta}_q$ and distance metric $\mathbf{D}_q$ of one local model centered at the query point $\mathbf{c}_q = \mathbf{x}_q$.

Alternatively, in the "memoryless" approach, multiple kernels are created as needed to cover the input space, and the sufficient statistics of the weighted regression are updated incrementally with recursive ▶ least squares (Schaal and Atkeson 1998). This approach does not require storage of data points in memory. Predictions of neighboring local models can be blended, improving function fitting results in the spirit of committee machines.

We describe some algorithms of both flavors next.

Memory-Based Locally Weighted Regression (LWR)

The original locally weighted regression algorithm was introduced by Cleveland (1979) and popularized in the machine learning and learning control community by Atkeson (1989). The algorithm – categorized as a "lazy" approach – can be summarized as follows below (for algorithmic pseudo-code, see Schaal et al. 2002):

- All training data is collected in the rows of the matrix $\mathbf{X}$ and the vector (For simplicity, only functions with a scalar output are addressed. Vector-valued outputs can be learned either by fitting a separate learning system for each output or by modifying the algorithms to fit multiple outputs (similar to multi-output linear regression).) **t**.
- For every query point $\mathbf{x}_q$, the weighting kernel is centered at the query point.

- The weights are computed with Eq. (4), and all data points' weights are collected in the diagonal weight matrix $\mathbf{W}_q$
- The local regression coefficients are computed as

$$\boldsymbol{\beta}_q = \left(\mathbf{X}^T\mathbf{W}_q\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{W}_q\mathbf{t} \qquad (5)$$

- A prediction is formed with $y_q = \left[\mathbf{x}_q^T\ 1\right]\boldsymbol{\beta}_q$.

As in all kernel methods, it is important to optimize the kernel parameters in order to get optimal function fitting quality. For LWR, the critical parameter determining the ▶ bias-variance tradeoff is the distance metric $\mathbf{D}_q$. If the kernel is too narrow, it starts fitting noise. If it is too broad, oversmoothing will occur. $\mathbf{D}_q$ can be optimized with leave-one-out cross validation to obtain a *globally* optimal value, i.e., the same $\mathbf{D}_q = \mathbf{D}$ is used throughout the entire input space of the data. Alternatively, $\mathbf{D}_q$ can be *locally* optimized as a function of the query point, i.e., obtain a $\mathbf{D}_q$ (as indicated by the subscript "q"). In the recent machine learning literature (in particular, work related to kernel methods), such input-dependent kernels are referred to as nonstationary kernels.

Locally Weighted Projection Regression (LWPR)

Schaal and Atkeson (1998) suggested a memoryless version of LWR, called RFWR, in order to avoid the expensive ▶ nearest neighbor computations—particularly for large training data sets—of LWR and to have fast real-time (In most robotic systems, "real time" means on the order of maximally 1–10 ms computation time, corresponding to a 1000 to 100 Hz control loop.) prediction performance. The main ideas of the RFWR algorithm (Schaal and Atkeson 1998) are listed below:

- Create new kernels only if no existing kernel in memory covers a training point with some minimal activation weight.
- Keep all created kernels in memory and update the weighted regression with weighted recursive ▶ least squares for new training points $\{\mathbf{x}, t\}$:

$$\boldsymbol{\beta}_k^{n+1} = \boldsymbol{\beta}_k^n + w\mathbf{P}^{n+1}\tilde{\mathbf{x}}\left(t - \tilde{\mathbf{x}}^T\boldsymbol{\beta}_k^n\right)$$

$$\text{where } \mathbf{P}_k^{n+1} = \frac{1}{\lambda}\left(\mathbf{P}_k^n - \frac{\mathbf{P}_k^n\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T\mathbf{P}_k^n}{\frac{\lambda}{w} + \tilde{\mathbf{x}}^T\mathbf{P}_k^n\tilde{\mathbf{x}}}\right) \text{ and } \tilde{\mathbf{x}} = \left[\mathbf{x}^T\ 1\right]^T. \qquad (6)$$

- Adjust the distance metric $\mathbf{D}_q$ for each kernel with a gradient descent technique using leave-one-out cross validation.
- Make a prediction for a query point taking a weighted average of predictions from all local models:

$$\mathbf{y}_q = \frac{\sum_{k=1}^{K} w_{q,k}\hat{\mathbf{y}}_{q,k}}{\sum_{k=1}^{K} w_{q,k}} \qquad (7)$$

Adjusting the distance metric $\mathbf{D}_q$ with leave-one-out cross validation *without* keeping all training data in memory is possible due to the PRESS residual. The PRESS residual allows the leave-one-out cross validation error to be computed in closed form without needing to actually exclude a data point from the training data.

Another deficiency of LWR is its inability to scale well to high-dimensional input spaces since the ▶ covariance matrix inversion in Eq. (5) becomes severely ill-conditioned. Additionally, LWR becomes expensive to evaluate as the number of local models to be maintained increases. Vijayakumar et al. (2005) suggested local ▶ dimensionality reduction techniques to handle this problem. Partial least squares (PLS) regression is a useful ▶ dimensionality reduction method that is used in the LWPR algorithm (Vijayakumar et al. 2005). In contrast to PCA methods, PLS performs ▶ dimensionality reduction for ▶ regression, i.e., it eliminates subspaces of the input space that minimally correlates with the outputs, not just parts of the input space that have low variance.

While LWPR is typically used in conjunction with linear local models, the use of local nonparametric models, such as Gaussian processes, has also been explored (Nguyen-Tuong et al. 2008). Finally, LWPR is currently one of the best developed locally weighted regression algorithms for control (Klanke et al. 2008) and has been applied to learning control problems with over 100 input dimensions.

A Full Bayesian Treatment of Locally Weighted Regression

Ting et al. (2008) proposed a fully probabilistic treatment of LWR in an attempt to avoid cross-validation procedures and minimize any manual parameter tuning (e.g., gradient descent rates, kernel initialization, forgetting rates, etc.). The resulting Bayesian algorithm learns the distance metric of local linear model (For simplicity, a local linear model is assumed, although local polynomials can be used as well.) probabilistically, can cope with high input dimensions, and rejects data outliers automatically. The main ideas of Bayesian LWR are listed below (please see Ting 2009 for details):

- Introduce hidden variables $\mathbf{z}$ to the local linear model to decompose the statistical estimation problem into d individual estimation problems (where d is the number of input dimensions). The result is an iterative expectation-maximization (EM) algorithm that is of linear ▸ computational complexity in d and the number of training data samples N, i.e., $O(Nd)$.
- Associate a scalar weight w_i with each training data sample $\{\mathbf{x}_i, t_i\}$, placing a Bernoulli ▸ prior probability distribution over a weight w_{im} *for each input dimension* m so that the weights are positive and between 0 and 1:

$$w_i = \prod_{m=1}^{d} w_{im} \text{ where}$$

$$w_{im} \sim \text{Bernoulli}\left(q_{im}\right) \text{ for } i = 1, .., N; \quad m = 1, .., d \tag{8}$$

The weight w_i indicates a training sample's contribution to the local model. The formulation of the parameter q_{im} determines the shape of the weighting function applied to the local model. The weighting function q_{im} used in Bayesian LWR is listed below:

$$q_{im} = \frac{1}{1 + \left(x_{im} - x_{qm}\right)^2 h_m} \text{ for } i = 1, .., N; \quad m = 1, .., d \tag{9}$$

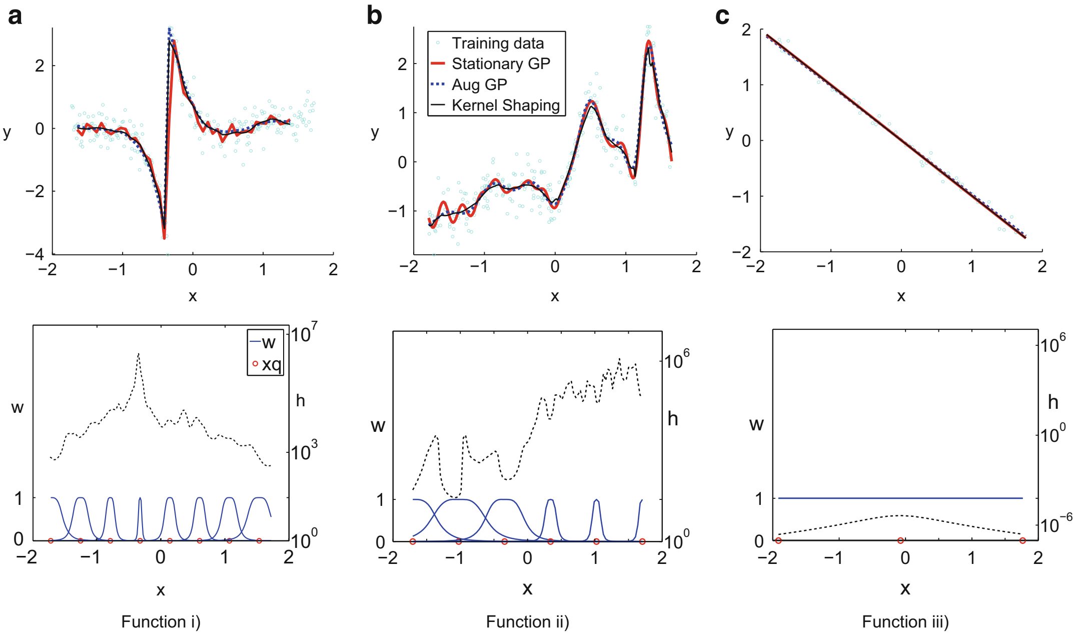
where $\mathbf{x}_q \in \Re^{d \times 1}$ is the query input point and h_m is the bandwidth parameter/distance metric of the local model in the m-th input dimension.
- Place a gamma ▸ prior probability distribution over the distance metric h_m:

$$h_m \sim \text{Gamma}\left(a_{hm0}, b_{hm0}\right) \tag{10}$$

where $\{a_{hm0}, b_{hm0}\}$ are the prior parameter values of the gamma distribution.
- Treat the model as an EM-like ▸ regression problem, using ▸ variational approximations to achieve analytically tractable inference of the ▸ posterior probability distributions.

This Bayesian method can also be applied as general kernel shaping algorithm for global ▸ kernel learning methods that are linear in the parameters (e.g., to realize nonstationary ▸ Gaussian processes (Ting et al. 2008), resulting in an augmented nonstationary ▸ Gaussian process).

Figure 2 illustrates Bayesian kernel shaping's bandwidth adaptation abilities on several synthetic data sets, comparing it to a stationary ▸ Gaussian process and the augmented nonstationary ▸ Gaussian process. For the ease of visualization, the following one-dimensional functions are considered: (i) a function with a discontinuity, (ii) a spatially inhomogeneous function, and (iii) a straight line function. Figure 2 shows the predicted outputs of all three models trained on noisy data drawn from data sets (i)–(iii). The local kernel shaping algorithm smoothens over regions where a stationary ▸ Gaussian process overfits, and yet, it still manages to capture regions of highly varying curvature, as seen in



Locally Weighted Regression for Control, Fig. 2 Predicted outputs using a stationary Gaussian process (GP), the augmented nonstationary GP, and local kernel shaping on three different data sets. Figures on the *bottom row* show the bandwidths learned by local kernel shaping and the corresponding weighting kernels (in *dotted black lines*) for various input query points (shown in *red circles*)

Fig. 2a, b. It correctly adjusts the bandwidths h with the curvature of the function. When the data looks linear, the algorithm opens up the weighting kernel so that all data samples are considered, as Fig. 2c shows.

From the viewpoint of ▶ learning control, ▶ overfitting—as seen in the ▶ Gaussian process in Fig. 2—can be detrimental since learning control often relies on extracting local linearizations to derive controllers. Obtaining the wrong sign on a slope in a local linearization may destabilize a controller.

In contrast to LWPR, the Bayesian LWR method is a "lazy" learner, although memoryless versions could be derived. Future work will also have to address how to incorporate ▶ dimensionality reduction methods for robustness in high dimensions. Nevertheless, it is a first step toward a probabilistic locally weighted regression method with minimal parameter tuning required by the user.

From Global to Local: Local Regression with Coupling Between Local Models

Meier et al. (2014) offer an alternative approach to local learning. They start out with the global objective (Eq. 3) and reformulate it to capture the idea of local models that cooperate to generate a function fit, resulting in

$$J = E\left[\frac{1}{2}\sum_{i=1}^{N}\left(\mathbf{t}_i - \sum_{k=1}^{K} w_{k,i}\left(\mathbf{x}_i^T \boldsymbol{\beta}_k\right)\right)^2\right]. \tag{11}$$

With this change, a local models' contribution $\hat{y}_k = \mathbf{x}_i^T \boldsymbol{\beta}_k$ toward the fit of target $\mathbf{t}_i$ is localized through weight $w_{k,i}$. However, this form of localization couples all local models. For efficient learning, local Gaussian regression (LGR) thus employs approximations to decouple learning of parameters. The main ideas of LGR are:

- Introduce Gaussian hidden variables $\mathbf{f}_k$ that form virtual targets for the weighted contribution of the kth local model:

$$f_{k,i} = \mathcal{N}\left(w_{k,i}(\mathbf{x}_i^T\boldsymbol{\beta}_k),\ \beta_m^{-1}\right) \tag{12}$$

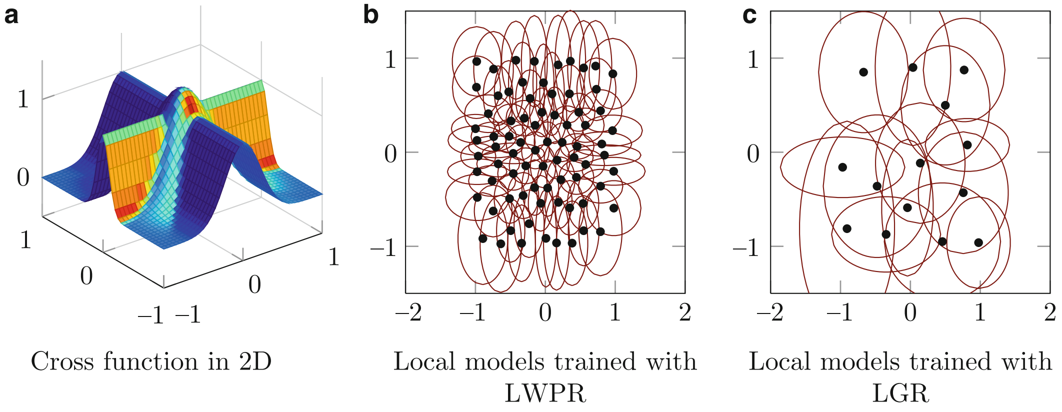
  Assume that the target t is observed with Gaussian noise and that the hidden variables f_k need to sum up to noisy target t_i

$$t_i = \mathcal{N}\left(\sum_k f_{k,i},\ \beta_y^{-1}\right) \tag{13}$$

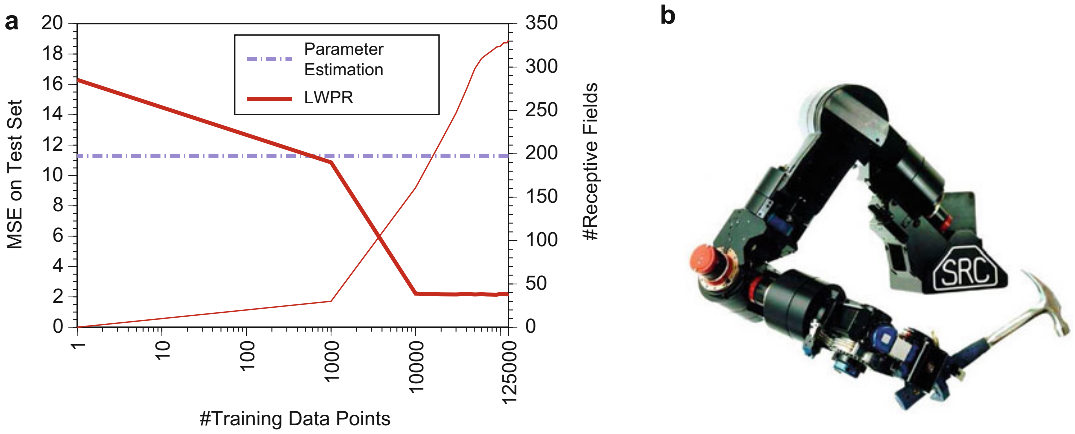
  In its exact form, this model learning procedure will couple all local models parameters.
- Employ a variational approximation to decouple local models. This results in an iterative (EM style) learning procedure, between updating posteriors over hidden variables $\mathbf{f}_k$ followed by posterior updates for regression parameters $\boldsymbol{\beta}_k$, for all local models $k = 1, \ldots, K$.
- The updates over the hidden variables $\mathbf{f}_k$ turn out to be a form of message passing between local model predictions. This step allows the redistribution of virtual target values for each local model. This *communication* between local models is what distinguishes LGR from typical LWR approaches. This update is linear in the number of local models and in the number of data points.
- The parameter updates ($\boldsymbol{\beta}_k$ and $\mathbf{D}_k$) per local model become completely independent through the variational approximation, resulting in a localized learning algorithm, similar in spirit to LWR.
- Place Gaussian priors over regression parameters $\boldsymbol{\beta}_k \sim \mathcal{N}(\boldsymbol{\beta}_k; 0, \text{diag}\,(\boldsymbol{\alpha}_k))$ that allow for automatic relevance determination of the input dimensions.
- For incrementally incoming data, apply recursive Bayesian updates that utilize the posterior over parameters at time step $t-1$ to be the prior over parameters at time step t. Furthermore, new local models are added if no existing local model is activated with some minimal activation weight, similar to LWPR.
- Prediction for a query input $\mathbf{x}_q$ becomes a weighted average of local models predictions

$$y_q = \sum_{k=1}^{K} w_{k,q}(\mathbf{x}_q^T\boldsymbol{\beta}_k)$$

More details and a pseudo-algorithm for incremental LGR can be found in Meier et al. (2014). Figure 3 illustrates the different shapes of local models being learned by LWPR and LGR. Local models learned by LGR *collaborate* to generate a good fit, as visualized in Fig. 3c. Compared to



Locally Weighted Regression for Control, Fig. 3 Local models trained on data from the 2D cross function for LWPR and LGR. Local models trained via LWPR (visualized in (**b**)) do not know of each other, while local models trained by LGR (visualized in (**c**)) *collaborate* to generate a function fit



Locally Weighted Regression for Control, Fig. 4 Learning an inverse dynamics model in real time with a high-performance anthropomorphic robot arm. (**a**) Learning curve LWPR online learning. (**b**) Seven degree-of-freedom Sarcos robot arm

LWPR, this often allows LGR to achieve similar predictive performance while using fewer local models.

Finally, an interesting structural feature of local ▶ Gaussian regression is that it easily extends to a model with finitely many local nonparametric ▶ Gaussian process models.

Applications

Learning Internal Models with LWPR

Learning an internal model is one of most typical applications of local regression methods for control. The model could be a forward model (e.g., the nonlinear differential equations of robot dynamics), an inverse model (e.g., the equations that predict the amount of torque to achieve a change of state in a robot), or any other function that models associations between input and output data about the environment. The models are used, subsequently, to compute a controller, e.g., an inverse dynamics controller similar to Eq. (16). Models for complex robots such as like humanoids exceed easily a hundred input dimensions. In such high-dimensional spaces, it is hopeless to assume that a representative data set can be collected for offline training that can generalize sufficiently to related tasks. Thus, the local regression philosophy involves having a learning algorithm that can learn rapidly when entering a new part of the state space such that it can achieve acceptable ▶ generalization performance almost instantaneously. Both LWPR (Vijayakumar et al. 2005) and incremental LGR (Meier et al. 2014) have been applied to inverse dynamics learning tasks.

Figure 4 demonstrates ▶ online learning of an inverse dynamics model for the elbow joint (cf. Eq. 16) for a Sarcos Dexterous Robot Arm. The robot starts with no knowledge about this model, and it tracks some randomly varying desired trajectories with a proportional-derivative (PD) controller. During its movements, training data consisting of tuples $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \tau)$—which model a mapping from joint position, joint velocities, and joint accelerations $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ to motor torques τ—are collected (at about every 2 ms). Here, every data point is used to train a LWPR function approximator, which generates a feedforward command for the controller. The learning curve is shown in Fig. 4a.

Using a test set created beforehand, the model predictions of LWPR are compared every 1000 training points with that of a parameter estimation method. The parameter estimation approach fits the minimal number of parameters to an analytical model of the robot dynamics under an idealized rigid body dynamics (RBD) assumptions, using all training data (i.e., not incrementally). Given that the Sarcos robot is a hydraulic robot, the RBD assumption is not very suitable, and, as Fig. 4a shows, LWPR (in thick red line) outperforms the analytical model (in dotted blue line) after a rather short amount of training. After about 5 min of training (about 125,000 data points), very good performance is achieved, using about 350 local models. This example demonstrates (i) the quality of function approximation that can be achieved with LWPR and (ii) the online allocation of more local models as needed.

Learning Paired Inverse-Forward Models

Learning inverse models (such as inverse kinematics and inverse dynamics models) can be challenging since the inverse model problem is often a relation, not a function, with a one-to-many mapping. Applying any arbitrary nonlinear function approximation method to the inverse model problem can lead to unpredictably bad performance, as the training data can form non-convex solution spaces, in which averaging is inappropriate. Architectures such as ▶ mixture models (in particular, mixture density networks) have been proposed to address problems with non-convex solution spaces. A particularly interesting approach in control involves learning linearizations of a forward model (which is proper function) and learning an inverse mapping within the local region of the forward model.

Ting et al. (2008) demonstrated such a forward-inverse model learning approach with Bayesian LWR to learn an inverse kinematics model for a haptic robot arm (shown in Fig. 5) in order to control the end effector along a desired trajectory in task space. Training data was collected while the arm performed random sinusoidal movements within a constrained box volume of Cartesian space. Each sample consists of the arm's joint angles $\mathbf{q}$, joint velocities $\dot{\mathbf{q}}$, end-effector position in Cartesian space $\mathbf{x}$, and end-effector velocities $\dot{\mathbf{x}}$. From this data, a forward kinematics model is learned:

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \tag{14}$$

Locally Weighted Regression for Control, Fig. 5 SensAble Phantom haptic robotic arm

where $\mathbf{J}(\mathbf{q})$ is the Jacobian matrix. The transformation from $\dot{\mathbf{q}}$ to $\dot{\mathbf{x}}$ can be assumed to be locally linear at a particular configuration $\mathbf{q}$ of the robot arm. Bayesian LWR is used to learn the forward model, and, as in LWPR, local models are only added if a training point is not already sufficiently covered by an existing local model. Importantly, the kernel functions in LWR are localized only with respect to $\mathbf{q}$, while the regression of each model is trained only on a mapping from $\dot{\mathbf{q}}$ to $\dot{\mathbf{x}}$—these geometric insights are easily incorporated as ▶ priors in Bayesian LWR, as they are natural to locally linear models. Incorporating these ▶ priors in other function approximators, e.g., ▶ Gaussian process regression, is not straightforward.

The goal of the robot task is to track a desired trajectory $(\mathbf{x}, \dot{\mathbf{x}})$ specified only in terms of x and z positions and velocities, i.e., the movement is supposed to be in a vertical plane in front of the robot, but the exact position of the vertical plane is not given. Thus, the task has one degree of redundancy, and the learning system needs to generate a mapping from $\{\mathbf{x}, \dot{\mathbf{x}}\}$ to $\dot{\mathbf{q}}$. Analytically, the inverse kinematics equation is

$$\dot{\mathbf{q}} = \mathbf{J}^{\#}(\mathbf{q})\dot{\mathbf{x}} - \alpha(\mathbf{I} - \mathbf{J}^{\#}\mathbf{J})\frac{\partial g}{\partial \mathbf{q}} \tag{15}$$

where $J^{\#}(\mathbf{q})$ is the pseudo-inverse of the Jacobian. The second term is a gradient descent optimization term for redundancy resolution, specified here by a cost function g in terms of joint angles $\mathbf{q}$.

To learn an inverse kinematics model, the local regions of $\mathbf{q}$ from the forward model can be reused since any inverse of $\mathbf{J}$ is locally linear within these regions. Moreover, for locally linear models, all solution spaces for the inverse model are locally convex, such that an inverse can be learned without problems. The redundancy issue can be solved by applying an additional weight to each data point according to a reward function. Since the experimental task is specified in terms of $\{\dot{x}, \dot{z}\}$, a reward is defined, based on a desired y coordinate, y_{des}, and enforced as a soft constraint. The resulting reward function is $g = e^{-\frac{1}{2}h(k(y_{des}-y)-\dot{y})^2}$, where k is a gain and h specifies the steepness of the reward. This ensures that the learned inverse model chooses a solution that pushes $\dot{y}$ toward y_{des}. Each forward local model is inverted using a weighted ▶ linear regression, where each data point is weighted by the kernel weight from the forward model and additionally weighted by the reward. Thus, a piecewise locally linear solution to the inverse problem can be learned efficiently.

Figure 6 shows the performance of the learned inverse model (Learnt IK) in a figure-eight tracking task. The learned model performs as well as the analytical inverse kinematics solution (Analytical IK), with root-mean-squared tracking errors in positions and velocities very close to that of the analytical solution.

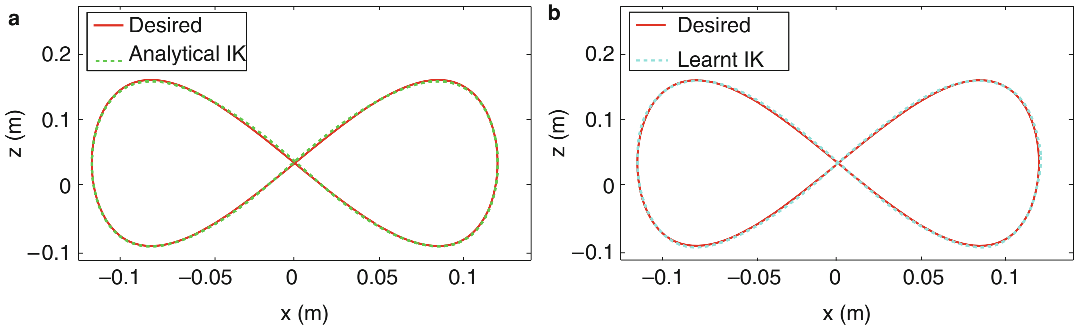
Learning Trajectory Optimizations

Mitrovic et al. (2008) have explored a theory for sensorimotor adaptation in humans, i.e., how humans replan their movement trajectories in the presence of perturbations. They rely on the iterative Linear Quadratic Gaussian (iLQG) algorithm (Todorov and Li 2004) to deal with the nonlinear and changing plant dynamics that may result from altered morphology, wear and tear, or external perturbations. They take advantage of the "on-the-fly" adaptation of locally weighted regression methods like LWPR to learn the forward dynamics of a simulated arm for the purpose of optimizing a movement trajectory between a start point and an end point.

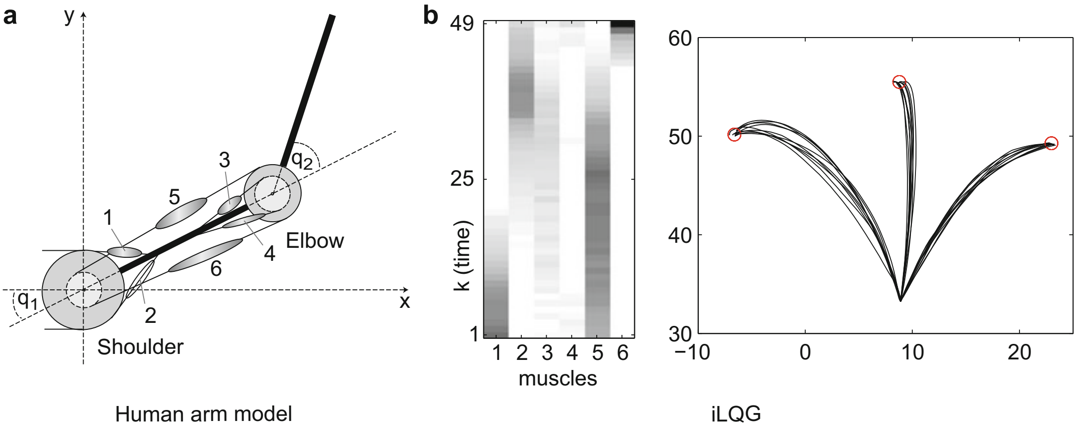
Figure 7a shows the diagram of a two degrees-of-freedom planar human arm model, which is actuated by four single-joint and two double-joint antagonistic muscles. Although kinematically simple, the system is over-actuated and, therefore, an interesting test bed because large redundancies in the dynamics have to be resolved. The dimensionality of the control signals makes adaptation processes (e.g., to external force fields) quite demanding.

The dynamics of the arm is, in part, based on standard RBD equations of motion:

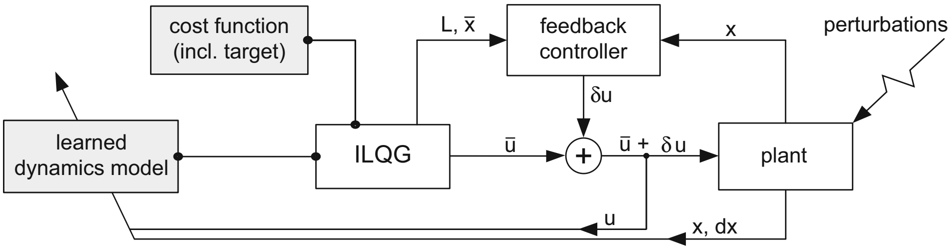
$$\tau = \mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} \qquad (16)$$



Locally Weighted Regression for Control, Fig. 6 Desired versus actual trajectories for SensAble Phantom robot arm. (**a**) Analytical solution. (**b**) Learned solution



Locally Weighted Regression for Control, Fig. 7 (**a**) Human arm model with six muscles; (**b**) Optimized control sequence (*left*) and resulting trajectories (*right*) using the known analytic dynamics model. The control sequences (left target only) for each muscle (1–6) are drawn from bottom to top, with *darker gray* levels indicating stronger muscle activation

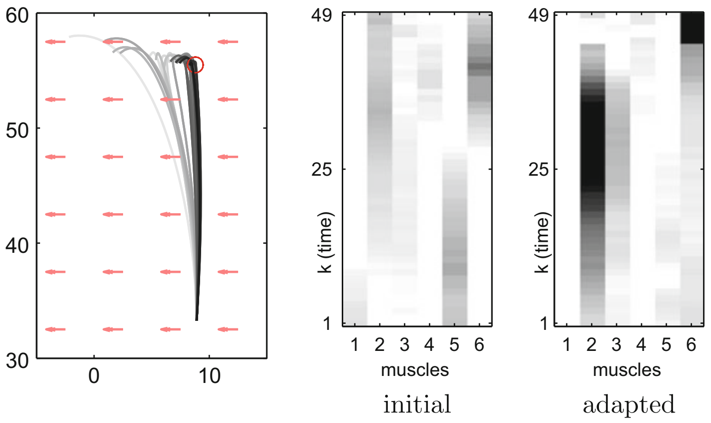


Locally Weighted Regression for Control, Fig. 8 Illustration of learning and control scheme of the iterative Linear Quadratic Gaussian (iLQG) algorithm with learned dynamics

where τ are the joint torques; $\mathbf{q}$ and $\dot{\mathbf{q}}$ are the joint angles and velocities, respectively; $\mathbf{M}(\mathbf{q})$ is the two-dimensional symmetric joint space inertia matrix; and $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ accounts for Coriolis and centripetal forces. Given the antagonistic muscle-based actuation, it is not possible to command joint torques directly. Instead, the effective torques from the muscle activations $\mathbf{u}$—which happens to be quadratic in $\mathbf{u}$—should be used. As a result, in contrast to standard torque-controlled robots, the dynamics equation in Eq. (16) is *non-linear in the control signals* $\mathbf{u}$.

The iLQG algorithm (Todorov and Li 2004) is used to calculate solutions to "localized" linear and quadratic approximations, which are iterated to improve the global control solution. However, it relies on an analytical forward dynamics model $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ and finite difference methods to compute gradients. To alleviate this requirement and to make iLQG adaptive, LWPR can be used to learn an approximation of the plant's forward dynamics model. Figure 8 shows the control diagram, where the "learned dynamics model" (the forward model learned by LWPR) is then updated in an ▸ online fashion with every iteration to cope with changes in dynamics. The resulting framework is called iLQG-LD (iLQG with learned dynamics).

Movements of the arm model in Fig. 7a are studied for fixed time horizon reaching movement. The manipulator starts at an initial position $\mathbf{q}_0$ and reaches toward a target $\mathbf{q}_{tar}$. The cost function to be optimized during the movement is a combination of target accuracy and amount of muscle activation (i.e., energy consumption). Figure 7b shows trajectories of generated movements for three reference targets (shown in red circles) using the feedback controller from iLQG with the analytical plant dynamics. The trajectories generated with iLQG-LD (where the forward



Locally Weighted Regression for Control, Fig. 9 Adaptation to a unidirectional constant force field (indicated by the *arrows*). *Darker lines* indicate better trained models. In particular, the left-most trajectory corresponds to the "initial" control sequence, which was calculated using the LWPR model *before* the adaptation process. The fully "adapted" control sequence results in a nearly straight line reaching movement

plant dynamics are learned with LWPR) are omitted as they are hardly distinguishable from the analytical solution.

A major advantage of iLQG-LD is that it does not rely on an accurate analytic dynamics model; this enables the framework to predict adaptation behavior under an ideal observer planning model. Reaching movements were studied where a constant unidirectional force field acting perpendicular to the reaching movement was generated as a perturbation (see Fig. 9 (left)). Using the iLQG-LD model, the manipulator gets strongly deflected when reaching for the target because the learned dynamics model cannot yet account for the "spurious" forces. However, when the deflected trajectory is used as training data and the dynamics model is updated ▸ online, the tracking improves with each new successive trial (Fig. 9 (left)). Please refer to Mitrovic et al. (2008) for more details. Aftereffects upon removing the force field, very similar to those observed in human experiments, are also observed.

Cross-References

- ▸ Bias Variance Decomposition
- ▸ Dimensionality Reduction
- ▸ Direct Controller
- ▸ Incremental Learning
- ▸ Kernel Methods
- ▸ Lazy Learning
- ▸ Linear Regression
- ▸ Mixture Model
- ▸ Online Learning
- ▸ Overfitting
- ▸ Radial Basis Function Networks
- ▸ Regression
- ▸ Supervised Learning
- ▸ Value Function Approximation

Programs and Data

http://www-clmc.usc.edu/software
http://www.ipab.inf.ed.ac.uk/slmc/software/

Recommended Reading

Aström KJ, Wittenmark B (1989) Adaptive control. Addison-Wesley, Reading

Atkeson C (1989) Using local models to control movement. In: Proceedings of the advances in neural information processing systems, vol 1. Morgan Kaufmann, San Mateo, pp 157–183

Atkeson C, Moore A, Schaal S (1997) Locally weighted learning. AI Rev 11:11–73

Cleveland WS (1979) Robust locally weighted regression and smoothing scatterplots. J Am Stat Assoc 74:829–836

Hastie T, Loader C (1993) Local regression: automatic kernel carpentry. Stat Sci 8:120–143

Jordan MI, Jacobs R (1994) Hierarchical mixtures of experts and the EM algorithm. Neural Comput 6:181–214
Klanke S, Vijayakumar S, Schaal S (2008) A library for locally weighted projection regression. J Mach Learn Res 9:623–626
Meier F, Hennig P, Schaal S (2014) Incremental local Gaussian regression. In: Proceedings of advances in neural information processing systems, Montreal, vol 27
Mitrovic D, Klanke S, Vijayakumar S (2008) Adaptive optimal control for redundantly actuated arms. In: Proceedings of the 10th international conference on the simulation of adaptive behavior, Osaka. Springer, pp 93–102
Nguyen-Tuong D, Peters J, Seeger M (2008) Local Gaussian process regression for real-time online model learning. In: Proceedings of advances in neural information processing systems, Vancouver, vol 21
Schaal S, Atkeson CG (1998) Constructive incremental learning from only local information Neural Comput 10(8):2047–2084
Schaal S, Atkeson CG, Vijayakumar S (2002) Scalable techniques from nonparametric statistics. Appl Intell 17:49–60
Ting J (2009) Bayesian methods for autonomous learning systems. Phd Thesis, Department of Computer Science, University of Southern California
Ting J, Kalakrishnan M, Vijayakumar S, Schaal S (2008) Bayesian kernel shaping for learning control. In: Proceedings of advances in neural information processing systems, Vancouver, vol 21. MIT Press, pp 1673–1680
Todorov E, Li W (2004) A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In: Proceedings of 1st international conference of informatics in control, automation and robotics, Setúbal
Vijayakumar S, D'Souza A, Schaal S (2005) Incremental online learning in high dimensions. Neural Comput 17:2602–2634

Logic of Generality

Luc De Raedt
Department of Computer Science, Katholieke
Universiteit Leuven, Heverlee, Leuven, Belgium

Synonyms

Generality and logic; Induction as inverted deduction; Inductive inference rules; Is more general than; Is more specific than; Specialization

Definition

One hypothesis is *more general* than another one if it covers all instances that are also covered by the latter one. The former hypothesis is called a ▸ *generalization* of the latter one, and the latter a ▸ *specialization* of the former. When using logical formulae as hypotheses, the generality relation coincides with the notion of logical entailment, which implies that the generality relation can be analyzed from a logical perspective. The logical analysis of generality, which is pursued in this chapter, leads to the perspective of *induction as the inverse of deduction*. This forms the basis for an analysis of various logical frameworks for reasoning about generality and for traversing the space of possible hypotheses. Many of these frameworks (such as for instance, *θ-subsumption*) are employed in the field of ▸ inductive logic programming and are introduced below.

Motivation and Background

Symbolic machine learning methods typically learn by searching a hypothesis space. The hypothesis space can be (partially) ordered by the ▸ generality relation, which serves as the basis for defining operators to traverse the space as well as for pruning away unpromising parts of the search space. This is often realized through the use of refinement operators, that is, generalization and specialization operators. Because many learning methods employ a ▸ hypothesis language that is logical or that can be reformulated in logic, it is interesting to analyze the generality relation from a logical perspective. When using logical formulae as hypotheses, the generality relation closely corresponds to logical entailment. This allows us to directly transfer results from logic to a machine learning context. In particular, machine learning operators can be derived from logical inference rules. The logical theory of generality provides a framework for transferring these results. Within the standard setting of inductive logic programming, learning from entailment, specialization is realized through deduction, and generalization through induction, which is considered to be the inverse of deduction. Different deductive inference rules lead to different frameworks for generalization and specialization. The most popular one is that of θ-subsumption, which is employed by the vast majority of contemporary inductive logic programming systems.

Theory

A hypothesis g is *more general than* a hypothesis s if and only if g covers all instances that are also covered by s, more formally, if covers$(s) \subseteq$ covers(g), in which case, covers(h) denotes the set of all instances covered by the hypothesis h.

There are several possible ways to represent hypotheses and instances in logic (De Raedt 2008, 1997), each of which results in a different setting with a corresponding covers relation. Some of the best known settings are *learning from entailment*, learning from interpretations, and learning from proofs.

Learning from Entailment

In learning from entailment, both hypotheses and instances are logical formulae, typically *definite clauses*, which underlie the programming language Prolog (Flach 1994). Furthermore, when learning from entailment, a hypothesis h covers an instance e if and only if $h \models e$, that is, when h logically entails e, or equivalently, when e is a logical consequence of h. For instance, consider the hypothesis h:

```
  flies :- bird, normal.
  bird :- blackbird.
  bird :- ostrich.
```

The first clause or rule can be read as `flies` *if* `normal` *and* `bird`, that is, normal birds fly. The second and third states that blackbirds are birds. Consider now the examples e_1:

```
flies :- blackbird, normal, small.
```

and e_2:

```
flies :- ostrich, small.
```

Example e_1 is covered by h, because it is a logical consequence of h, that is, $h \models e_1$. On the other hand, example e_2 is not covered, which we denote as $h \not\models e_2$.

When learning from entailment, the following property holds:

Property 1 A hypothesis g is more general than a hypothesis s if and only if g logically entails s, that is, $g \models s$.

This is easy to see. Indeed, g is more general than s if and only if covers$(s) \subseteq$ covers(g) if and only if for all examples $e : (s \models e) \rightarrow (g \models e)$, if and only if $g \models s$. For instance, consider the hypothesis h_1:

```
flies :- blackbird, normal.
```

Because $h \models h_1$, it follows that h covers all examples covered by h_1, and hence, h generalizes h_1.

Property 1 states that the generality relation coincides with logical entailment when learning from entailment. In other learning settings, such as when *learning from interpretations*, this relationship also holds though the direction of the relationship might change.

Learning from Interpretations

In learning from interpretations, hypotheses are logical formulae, typically sets of definite clauses, and instances are interpretations. For propositional theories, interpretations are assignments of truth-values to propositional variables. For instance, continuing the `flies` illustration, two interpretations could be

{`blackbird, bird, normal, flies`} and {`ostrich, small`}

where we specify interpretations through the set of propositional variables that are true. An interpretation specifies a kind of possible world. A hypothesis h then covers an interpretation if and only if the interpretation is a model for the hypothesis. An interpretation is a model for a hypothesis if it satisfies all clauses in the hypothesis. In our illustration, the first interpretation is a model for the theory h, but the second is not. Because the condition part of the rule `bird :- ostrich.` is satisfied in the second interpretation (as it contains ostrich), the conclusion part, that is, bird, should also belong to the interpretation in order to have a model. Thus, the first example is covered by the theory h, but the second is not.

When learning from interpretations, a hypothesis g is more general than a hypothesis s if and only if for all examples e: (e is a model of s) → (e is a model of g), if and only if $s \models g$.

Because the learning from entailment setting is more popular than the learning from interpretations setting, we shall employ in this section the usual convention that states that one hypothesis g is more general than a hypothesis s if and only if $g \models s$.

An Operational Perspective

Property 1 lies at the heart of the theory of *inductive logic programming* and generalization because it directly relates the central notions of logic with those of machine learning (Muggleton and De Raedt 1994). It is also extremely useful because it allows us to directly transfer results from logic to machine learning.

This can be illustrated using traditional deductive inference rules, which start from a set of formulae and derive a formula that is entailed by the original set. For instance, consider the resolution inference rule for propositional definite clauses:

$$\frac{h \leftarrow g, a_1, \ldots, a_n, \text{ and } g \leftarrow b_1, \ldots, b_m}{h \leftarrow b_1, \ldots, b_m, a_1, \ldots, a_n,}. \quad (1)$$

This inference rule starts from the two rules above the line and derives the so-called *resolvent* below the line. This rule can be used to infer h_1 from h. An alternative deductive inference rule adds a condition to a rule:

$$\frac{h \leftarrow a_1, \ldots, a_n}{h \leftarrow a, a_1, \ldots, a_n,} \quad (2)$$

This rule can be used to infer that h_1 is more general than the clause used in example e_1. In general, a deductive inference rule can be written as

$$\frac{g}{s}. \quad (3)$$

If s can be inferred from g and the operator is *sound*, then $g \models s$. Thus, applying a deductive inference rule realizes specialization, and hence, deductive inference rules can be used as specialization operators. A *specialization operator* maps a hypothesis onto a set of its specializations. Because specialization is the inverse of generalization, *generalization operators* – which map a hypothesis onto a set of its generalizations – can be obtained by inverting deductive inference rules. The inverse of a deductive inference rule written in format (3) works from bottom to top, that is, from s to g. Such an inverted deductive inference rule is called an *inductive* inference rule. This leads to the view of induction as the inverse of deduction. This view is operational as it implies that each deductive inference rule can be inverted into an inductive one, and, also, that each inference rule provides an alternative framework for generalization.

An example of a generalization operator is obtained by inverting the adding condition rule (2). It corresponds to the well-known "dropping condition" rule (Michalski 1983). As will be seen soon, it is also possible to invert the resolution principle (1).

Before deploying inference rules, it is necessary to determine their properties. Two desirable properties are *soundness* and *completeness*. These properties are based on the repeated application of inference rules. Therefore, we write $g \vdash rs$ when there exists a sequence of hypotheses $h_1, \ldots, h_n$ such that

$$\frac{g}{h_1}, \frac{h_1}{h_2}, \ldots, \frac{h_n}{s} \text{ using } r. \quad (4)$$

A set of inference rules r is *sound* whenever $g \vdash rs$ implies $g \models s$; and *complete* whenever $g \models s$ implies $g \vdash rs$. In practice, soundness is always enforced though completeness is not always required in a machine learning setting. When working with incomplete rules, one should realize that the generality relation "$\vdash r$" is weaker than the logical one "$\models$."

The most important logical frameworks for reasoning about generality, such as θ-subsumption and resolution, are introduced below using the above introduced logical theory of generality.

Frameworks for Generality

Propositional Subsumption

Many propositional learning systems employ hypotheses that consist of rules, often definite clauses as in the `flies` illustration above. The propositional subsumption relation defines a generality relation among clauses and is defined through the adding condition rule (2). The properties follow from this inference rule by applying the logical theory of generalization presented above. More specifically, the generality relation $\vdash a$ induced by the adding condition rule states that a clause g is more general than a clause s, if s can be derived from g by adding a sequence of conditions to g. Observing that a clause $h \leftarrow a_1, \ldots, a_n$ is a disjunction of literals $h \vee \neg a_1 \vee \cdots \vee \neg a_n$ allows us to write it in set notation as $\{h, \neg a_1, \ldots, \neg a_n\}$. The soundness and completeness of propositional subsumption then follow from

$$g \vdash_a s \text{ if and only if } g \subseteq s \text{ if and only if } g \models s, \tag{5}$$

which also states that g subsumes s if and only if $g \subseteq s$.

The propositional subsumption relation induces a complete lattice on the space of possible clauses. A *complete lattice* is a partial order – a reflexive, antisymmetric, and transitive relation – where every two elements posses a unique least upper and greatest lower bound. An example lattice for rules defining the predicate `flies` in terms of `bird`, `normal`, and `small` is illustrated in the Hasse diagram depicted in Fig. 1.

The Hasse diagram also visualizes the different operators that can be used. The *generalization* operator ρ_g maps a clause to the set of its parents in the diagram, whereas the *specialization* operator ρ_s maps a clause to the set of its children. So far, we have defined such operators *implicitly* through their corresponding inference rules. In the literature, they are often defined *explicitly*:

$$\rho_g(h \leftarrow a_1, \ldots a_n)$$
$$= \{h \leftarrow a_1, \ldots a_{i-1}, a_{i+1}, \ldots, a_n | i = 1, \ldots, n\}. \tag{6}$$

In addition to using the inference rules directly, some systems such as Golem (Muggleton and Feng 1990) also exploit the properties of the underlying lattice by computing the least upper bound of two formulae. The least upper bound operator is known under the name of least general generalization (lgg) in the machine learning literature. It returns the least common ancestor in the Hasse diagram. Using a set notation for clauses, the definition of the lgg is:

$$\text{lgg}(c_1, c_2) = c_1 \cap c_2. \tag{7}$$

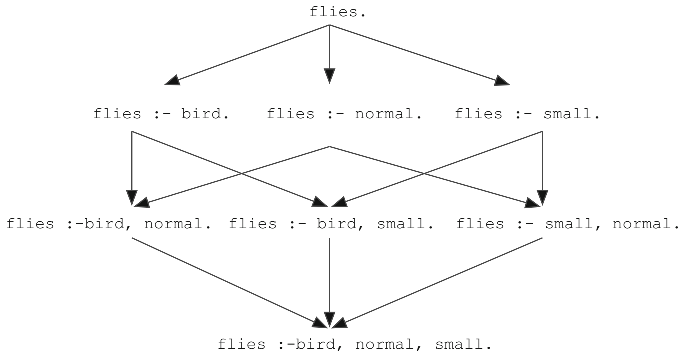
The least general generalization operator is used by machine learning systems that follow a cautious generalization strategy. They take two clauses corresponding to positive examples and minimally generalize them.

θ-Subsumption

The most popular framework for generality within inductive logic programming is θ-subsumption (Plotkin 1970). It provides a generalization relation for clausal logic and it extends propositional subsumption to first order logic.

A *definite clause* is an expression of the form $h \leftarrow a_1, \ldots, a_n$ where h and the a_i are logical atoms. An *atom* is an expression of the form $p(t_1, \ldots, t_m)$ where p is a *predicate name* (or, the name of a relation) and the t_i are terms. A *term* is either a constant (denoting an object in the domain of discourse), a variable, or a structured term of the form $f(u_1, \ldots, u_k)$ where f is a functor symbol (denoting a function in the domain of discourse) and the u_i are terms, see Flach (1994) for more details. Consider for instance the clauses

```
likes(X,Y) :- neighbours(X,Y).
likes(X,husbandof(Y)) :- likes(X,Y).
likes(X,tom) :- neighbours(X,tom),
             male(X).
```



Logic of Generality, Fig. 1 The Hasse diagram for the predicate `flies`

The first clause states that `X likes Y` if `X` is a `neighbour` of `Y`. The second one that `X likes` the `husband` of `Y` if `X likes Y`. The third one that all `male neighbours` of `tom` like `tom`.

θ-Subsumption is based not only on the adding condition rule (2) but also on the *substitution rule*:

$$\frac{g}{g\theta}. \tag{8}$$

The substitution rule applies a substitution θ to the definite clause g. A substitution $\{V_1/t_1, \ldots, V_n/t_n\}$ is an assignment of terms to variables. Applying a substitution to a clause c yields the instantiated clause, where all variables are simultaneously replaced by their corresponding terms.

θ-subsumption is then the generality relation induced by the substitution and the adding condition rules. Denoting this set of inference rules by t, we obtain our definition of θ-subsumption:

$$\begin{aligned} g\theta\text{-subsumption}s &\text{ if and only if } g \vdash_t s \\ &\text{if and only if } \exists\theta : g\theta \subseteq s. \end{aligned} \tag{9}$$

For instance, the first clause for `likes` subsumes the third one with the substitution `{Y/tom}`.

- θ-subsumption has some interesting properties:
- θ-subsumption is sound.
- θ-subsumption is complete for clauses that are not self-recursive. It is incomplete for self-recursive clauses such as

```
nat(s(X)) :- nat(X)
nat(s(s(Y))) :- nat(Y)
```

  for which one can use resolution to prove that the first clause logically entails the second one, even though it does not θ-subsume it.
- Deciding θ-subsumption is an NP-complete problem.

Because θ-subsumption is relatively simple and decidable whereas logical entailment between single clauses is undecidable, it is used as the generality relation by the majority of inductive logic programming systems. These systems typically employ a specialization or refinement operator to traverse the search space. To guarantee systematic enumeration of the search space, the specialization operator ρ_s can be employed. $\rho_s(c)$ is obtained by applying the adding condition or substitution rule with the following restrictions.

- The adding condition rule only adds atoms of the form $p(V_1, \ldots, V_n)$, where the V_i are variables not yet occurring in the clause c.
- The substitution rule only employs *elementary substitutions*, which are of the form
  - $\{X \,/\, Y\}$, where X and Y are two variables appearing in c
  - $\{V \,/\, ct\}$, where V is a variable in c and ct a constant

- $\{V / f(V_1, \ldots, V_n)\}$, where V is a variable in c, f a functor of arity n and the V_i are variables not yet occurring in c.

A generalization operator can be obtained by inverting ρ_s, which requires one to invert substitutions. Inverting substitutions is not easy. While applying a substitution $\theta = \{\texttt{V/a}\}$ to a clause c replaces all occurrences of `V` by `a` and yields a unique clause $c\theta$, applying the substitution rule in the inverse direction does not necessarily yield a unique clause. If we assume the elementary substitution applied to c with

$$\frac{c}{q(a,a)}. \tag{10}$$

was $\{\texttt{V/a}\}$, then there are at least three possibilities for c: `q(a,V)`, `q(V,a)`, and `q(V,V)`.

θ-subsumption is reflexive, transitive but unfortunately not anti-symmetric, which can be seen by considering the clauses

```
parent(X,Y) :- father(X,Y).
parent(X,Y) :- father(X,Y),
               father(U,V).
```

The first clause clearly subsumes the second one because it is a subset. The second one subsumes the first with the substitution {`X/U`, `V/Y`}. The two clauses are therefore equivalent under θ-subsumption, and hence also logically equivalent. The loss of the anti-symmetry complicates the search process. The naive application of the specialization operator ρs may yield syntactic specializations that are logically equivalent. This is illustrated above where the second clause for `parent` is a refinement of the first one using the adding condition rule. In this way, useless clauses are generated, and if the resulting clauses are further refined, there is a danger that the search will end up in an infinite loop.

Plotkin (1970) has studied the quotient set induced by θ-subsumption and proven various interesting properties. The quotient set consists of classes of clauses that are equivalent under θ-subsumption. The class of clauses equivalent to a given clause c is denoted by

$$[c] = \{c' | c' \text{ is equivalent with } c \text{ under } \theta\text{-subsumption}\}. \tag{11}$$

Plotkin proved that

- The quotient set is well-defined w.r.t. θ-subsumption.
- There is a representative, a canonical form, of each equivalence class, the so-called *reduced clause*. The reduced clause of an equivalence class is the shortest clause belonging to class. It is unique up to variable renaming. For instance, in the `parent` example above, the first clause is in reduced form.
- The quotient set forms a complete lattice, which implies that there is a least general generalization of two equivalence classes. In the inductive logic programming literature, one often talks about the least general generalization of two clauses.

Several variants of θ-subsumption have been developed. One of the most important ones is that of *OI*-subsumption (Esposito, Laterza, Malerba, and Semeraro, 1996). For functor-free clauses, it modifies the substitution rule by disallowing substitutions that unify two variables or that substitute a variable by a constant already appearing in the clause. The advantage is that the resulting relation is anti-symmetric, which avoids some of the above mentioned problems with refinement operators. On the other hand, the minimally general generalization of two clauses is not necessary unique, and hence, there exists no least general generalization operator.

Inverse Resolution

Applying resolution is a sound deductive inference rule and therefore realizes specialization. Reversing it yields inductive inference rules or generalization operators (Muggleton 1987; Muggleton and Buntine 1988). This is typically realized by combining the resolution principle with a copy operator. The resulting rules are called *absorption* (12) and *identification* (13). They start from the clauses below and induce the clause above the line. They are shown here only for the propositional case, as the first order case requires one to deal with substitutions as well as inverse substitutions.

$$\frac{h \leftarrow g, a_1, \ldots, a_n, \text{ and } g \leftarrow b_1, \ldots, b_m}{h \leftarrow b_1, \ldots, b_m, a_1, \ldots, a_n, \text{ and } g \leftarrow b_1, \ldots, b_m}. \tag{12}$$

$$\frac{h \leftarrow g, a_1, \ldots, a_n, \text{ and } g \leftarrow b_1, \ldots, b_m}{h \leftarrow b_1, \ldots, b_m, a_1, \ldots, a_n, \text{ and } h \leftarrow g, a_1, \ldots, a_n}. \tag{13}$$

Other interesting inverse resolution operators perform predicate invention, that is, they introduce new predicates that were not yet present in the original data. These operators invert two resolution steps. One such operator is the *intra-construction* operator (14). Applying this operator from bottom to top introduces the new predicate q that was not present before.

$$\frac{q \leftarrow l_1, \ldots, l_k, \text{ and } p \leftarrow k_1, \ldots, k_n, q \text{ and } q \leftarrow l'_1, \ldots, l'_m}{p \leftarrow k_1, \ldots, k_n, l_1, \ldots, l_k \text{ and } q \leftarrow k_1, \ldots, k_n, l'_1, \ldots, l'_m}. \tag{14}$$

The idea of inverting the resolution operator is very appealing because it aims at inverting the most popular deductive inference operator, but is also rather complicated due to the nondeterminism and the need to invert substitutions. Due to these complications, there are only few systems that employ inverse resolution operators.

Background Knowledge

Inductive logic programming systems employ background knowledge during the learning process. Background knowledge typically takes the form of a set of clauses B, which is then used by the covers relation. When learning from entailment in the presence of background knowledge B an example e is covered by a hypothesis h if and only if $B \cup h \models e$. This notion of coverage is employed in most of the work on inductive logic programming. In the intial `flies` example, the two clauses defining `bird` would typically be considered background knowledge.

The incorporation of background knowledge in the induction process has resulted in the frameworks for generality *relative* to a background theory. More formally, a hypothesis g is more general than a hypothesis s relative to the background theory B if and only if $B \cup g \models s$. The only inference rules that deal with multiple clauses are those based on (inverse) resolution. The other frameworks can be extended to cope with this generality relation following the logical theory of generalization. Various frameworks have been developed along these lines. Some of the most important ones are relative subsumption (Plotkin 1971) and generalized subsumption (Buntine 1998), which extend θ-subsumption and the notion of least general generalization toward the use of background knowledge. Computing the least general generalization of two clauses relative to the background theory is realized by first computing the most specific clauses covering the examples with regard to the background theory and then generalizing them using the least general generalization operator of θ-subsumption.

The first step is the most interesting one, and has been tackled under the name of *saturation* (Rouveirol 1994) and *bottom-clauses* (Muggleton 1995). We illustrate it within the framework of inverse entailment due to (Muggleton 1995). The bottom clause $\perp(c)$ of a clause c with regard to a background theory B is the most specific clause $\perp(c)$ such that

$$B \cup \perp(c) \models c. \tag{15}$$

If B consist of

```
polygon :- rectangle.
rectangle :- square.
oval :- circle.
```

and the example c is

```
positive :- red, square.
```

Then the bottom-clause $\bot\,(c)$ is

```
 positive :- red, rectangle,
                 square, polygon.
```

The bottom-clause is useful because it only lists those atoms that are relevant to the example, and only generalizations (under θ-subsumption) of $\bot\,(c)$ will cover the example. For instance, in the illustration, the bottom-clause mentions neither `oval` nor `circle` as clauses for `pos` containing these atoms will never cover the example clause c. Once the bottom-clause covering an example has been found the search process continues as if no background knowledge were present. Either specialization operators (typically under θ-subsumption) would search the space of clauses more general than $\bot\,(c)$, or the least general generalization of multiple bottom-clauses would be computed.

Equation (15) is equivalent to

$$B \cup \neg c \models \neg\bot(c). \tag{16}$$

which explains why the bottom-clause is computed by finding all factual consequences of $B \cup \neg c$ and then inverting the resulting clause again. On the example:

$\neg c = \{\neg$`positive, red, square`$\}$

and the set of all consequences is

$\neg\bot(\mathrm{c}) = \neg c \cup \{$`rectangle, polygon\verb`$\}$

which then yields $\bot(c)$ mentioned above. When dealing with first order logic, bottom-clauses can become infinite, and therefore, one typically imposes further restrictions on the atoms that appear in bottom-clauses. These restrictions are part of the *language bias*.

The textbook by Nienhuys-Cheng and de Wolf (1997) is the best reference for an in-depth formal description of various frameworks for generality in logic, in particular, for θ-subsumption and some of its variants. The book by De Raedt (2008) contains a more complete introduction to inductive logic programming and relational learning, and also introduces the key frameworks for generality in logic. An early survey of inductive logic programming and the logical theory of generality is contained in Muggleton and De Raedt (1994). Plotkin (1970, 1971) pioneered the use θ-subsumption and relative subsumption (under a background theory) for machine learning. Buntine (1998) extended these frameworks toward generalized subsumption, and Esposito et al. (1996) introduced *OI*-subsumption. Inverse resolution was first used in the system Marvin (Sammut and Banerji 1986), and then elaborated by Muggleton (1987) for propositional logic and by Muggleton and Buntine (1988) for definite clause logic. Various learning settings are studied by De Raedt (1997) and discussed extensively by De Raedt (2008). They are also relevant to probabilistic logic learning and ▸ statistical relational learning.

Recommended Reading

Buntine W (1998) Generalized subsumption and its application to induction and redundancy. Artif Intell 36:375–399

De Raedt L (1997) Logical settings for concept learning. Artif Intell 95:187–201

De Raedt L (2008) Logical and relational learning. Springer, New York

Flach PA (1994) Simply logical: intelligent reasoning by example. Wiley, New York

Michalski RS (1983) A theory and methodology of inductive learning. Artif Intell 20(2):111–161

Muggleton S (1987) Duce, an oracle based approach to constructive induction. In: Proceedings of the 10th international joint conference on artificial intelligence. Morgan Kaufmann, San Francisco, pp 287–292

Muggleton S (1995) Inverse entailment and Progol. New Gener Comput 13(3–4):245–286

Muggleton S, Buntine W (1988) Machine invention of first order predicates by inverting resolution. In: Proceedings of the 5th international workshop on machine learning. Morgan Kaufmann, San Francisco, pp 339–351

Muggleton S, De Raedt L (1994) Inductive logic programming: theory and methods. J Logic Program 19/20:629–679

Muggleton S, Feng C (1990) Efficient induction of logic programs. In: Proceedings of the 1st conference on algorithmic learning theory, Ohmsma, Tokyo, pp 368–381

Nienhuys-Cheng S-H, de Wolf R (1997) Foundations of inductive logic programming. Springer, Berlin
Plotkin GD (1970) A note on inductive generalization. In: Machine intelligence, vol 5. Edinburgh University Press, Edinburgh, pp 153–163
Plotkin GD (1971) A further note on inductive generalization. In: Machine intelligence, vol 6. Edinburgh University Press, Edinburgh, pp 101–124
Rouveirol C (1994) Flattening and saturation: two representation changes for generalization. Mach Learn 14(2):219–232
Sammut C, Banerji RB (1986) Learning concepts by asking questions. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning: an artificial intelligence approach, vol 2. Morgan Kaufmann, San Francisco, pp 167–192
Semeraro G, Esposito F, Malerba D (2006) Ideal refinement of datalog programs. In: Proceedings of the 5th international workshop on logic program synthesis and transformation, Utrecht. Lecture notes in computer science, vol 1048. Springer, pp 120–136

Logic Program

A logic program is a set of logical rules or ▶ clauses. Logic programs are employed to answer queries using the ▶ resolution inference rule. For example, consider the following logic program:

```
grandparent(X,Y) :- parent(X,Z),
                    parent(Z,Y).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
father(charles, william).
   mother(diana, william).
father(philip, charles).
   mother(elizabeth, charles).
father(john, diana).
   mother(frances, diana).
```

Using resolution we obtain the following answers to the query `:-grandparent(X,Y)`:

```
  X = philip,     Y = william ;
  X = john,       Y = william ;
  X = elizabeth,  Y = william ;
  X = frances,    Y = william.
```

Cross-References

▶ Clause
▶ First-Order Logic
▶ Prolog

Logical Consequence

▶ Entailment

Logical Regression Tree

▶ First-Order Regression Tree

Logistic Calibration

▶ Classifier Calibration

Logistic Regression

Synonyms

Logit model

Definition

Logistic regression provides a mechanism for applying the techniques of ▶ linear regression to ▶ classification problems. It utilizes a linear regression model of the form

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

where x_1 to x_n represent the values of the n attributes and β_0 to β_n represent weights. This model is mapped onto the interval [0,1] using

$$P(c_0|x_1 \ldots x_n) = \frac{1}{1 + e^{-z}}$$

where c_0 represents class 0.

Recommended Reading

Hastie T, Tibshirani R, Friedman J (2009) The elements of statistical learning, 2nd edn. Springer, New York

Logit Model

▶ Logistic Regression

Log-Linear Models

▶ Maximum Entropy Models for Natural Language Processing

Long-Term Potentiation of Synapses

By a suitable induction protocol, the connection between two neurons can be strengthened. If this change persists for hours, the effect is called a long-term potentation.

LOO Error

▶ Leave-One-Out Error

Loopy Belief Propagation

Loopy belief propagation is a heuristic inference algorithm for ▶ Bayesian networks. See ▶ Graphical Models for details.

Loss

Synonyms

Cost

Definition

The *cost* or *loss* of a prediction y', when the correct value is y, is a measure of the relative utility of that prediction given that correct value. A common loss function used with ▶ classification learning is ▶ zero-one loss. Zero-one loss assigns 0 to loss for a correct classification and 1 for an incorrect classification. ▶ Cost sensitive classification assigns different costs to different forms of misclassification. For example, misdiagnosing a patient as having appendicitis when he or she does not might be of lower cost than misdiagnosing the patient as not having it when he or she does. A common loss function used with ▶ regression is ▶ error squared. This is the square of the difference between the predicted and true values.

Loss Function

Synonyms

Cost function

Definition

A loss function is a function used to determine ▶ loss.

Lossy Compression

▶ Dimensionality Reduction

LVQ

▶ Learning Vector Quantization

LWPR

▶ Locally Weighted Regression for Control

LWR

▶ Locally Weighted Regression for Control

M

Machine Learning and Game Playing

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt,
Darmstadt, Deutschland
Department of Information Technology,
University of Leoben, Leoben, Austria

Abstract

Game playing is a major application area for research in artificial intelligence in general (Schaeffer and van den Herik 2002) and for machine learning in particular (Fürnkranz and Kubat 2001). Traditionally, the field is concerned with learning in strategy games such as tic-tac-toe (Michie 1963), checkers (▶ Samuel's checkers player), backgammon (▶ TD-Gammon), chess (Baxter et al. 2000; Björnsson and Marsland 2003; Donninger and Lorenz 2006; Sadikov and Bratko 2006), Go (Silver et al. 2016), Othello (Buro 2002), poker (Billings et al. 2002), or bridge (Amit and Markovitch 2006). However, recently computer and video games have received increased attention (Laird and van Lent 2001; Spronck et al. 2006; Ponsen et al. 2006).

Motivation and Background

Since the early days of the field, game-playing applications have been popular test beds for machine learning. This has several reasons:

- *Games allow to focus on intelligent reasoning.* Other components of intelligent agents, such as perception or physical actions, can be ignored.
- *Games are easily accessible.* A typical game-playing environment can be implemented within a few days, often hours. Exceptions are real-time computer games, for which only a few open-source test beds exist.
- *Games are very popular.* It is not very hard to describe the agent's task to the general public, and they can easily appreciate the achieved level of intelligence.

There are various types of problems that keep reoccurring in game-playing applications, for which solutions with machine learning methods are desirable, including opening book learning, learning of evaluation functions, player modeling, and others, which will be dealt with in the following.

Structure of the Learning System

Game-playing applications offer various challenges for machine learning. A wide variety of learning techniques have been used for tackling these problems. We cannot provide details on the learning algorithms here, but will instead focus on the problems and give some of the most relevant and most recent pointers to the literature. A more detailed survey can be found in Fürnkranz (2001).

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

Learning of Evaluation Functions

The most extensively studied learning problem in game playing is the automatic adjustment of the weights of an evaluation function. Typically, the situation is as follows: the game programmer has provided the program with a library of routines that compute important features of the current board position (e.g., the number of pieces of each kind on the board, the size of the territory controlled, etc.). What is not known is how to combine these pieces of knowledge and how to quantify their relative importance. Most frequently, these parameters are combined linearly, so that the learning task is to adjust the weights of a weighted sum. The main problem is that there are typically no direct target values that could be used as training signals. Exceptions are games or endgames that have been solved completely, which are treated further below. However, in general, algorithms use ▸ preference learning (where pairs of moves or positions are labeled according to which one is preferred by an expert player) or ▸ reinforcement learning (where moves or positions are trained based on information about the eventual outcome of the game) for tuning the evaluation functions.

The key problem with reinforcement learning approaches is the ▸ credit assignment problem, i.e., even though a game has been won (lost), there might be bad (good) moves in the game. Reinforcement learning takes a radical stance at this problem, giving all positions the same reinforcement signal, hoping that erroneous signals will be evened out over time. An early classic in this area is MENACE, a tic-tac-toe player that simulates reinforcement learning with delayed rewards (Michie 1963) using a stack of matchboxes, one for each position, each containing a number of beads in different colors, which represent the different legal moves in the position. Moves are selected by randomly drawing a bead out of the box that represents the current position. After a game is won, all played moves are reinforced by adding beads of the corresponding colors to these boxes; in the case of a lost game, corresponding beads are removed, thereby decreasing the probability that the same move will be played again.

The premier example of a system that has tuned its evaluation function to expert strength by playing millions of games against itself is the backgammon program ▸ TD-Gammon. Its key innovation was the use of a ▸ neural network instead of a position table, so that the reinforcement signal can be generalized to new unseen positions. Many authors have tried to copy TD-Gammon's learning methodology to other games (Ghory 2004). None of these successors, however, achieved a performance that was as impressive as TD-Gammon's. The reason for this seems to be that backgammon has various characteristics that make it perfectly suited for learning from self-play. Foremost, among these are the facts that the dice rolls guarantee sufficient variability, which allows to use training by self-play without the need for an explicit exploration/exploitation trade-off, and that it only requires a very limited amount of search, which allows to ignore the dependencies of search algorithm and search heuristic. These points have, for example, been addressed with limited success in the game of chess, where the program KnightCap (Baxter et al. 2000) integrates ▸ temporal difference learning into a game-tree search by using the final positions of the principal variation for updates and by using play on a game server for exploration.

Many aspects of evaluation function learning are still discussed in the current literature, including whether there are alternatives to reinforcement learning (e.g., evolutionary algorithms), which training strategies should be used (e.g., self-play vs. play against a teacher), etc. One of the key problems that has already been mentioned in ▸ Samuel's Checkers Player, namely, the automated construction of useful features, remains still largely unsolved. Some progress has, e.g., been made in the game of Othello, where a simple algorithm, very much like ▸ APriori, has been shown to produce valuable conjunctions of basic features (Buro 2002).

Learning Search Control

A more challenging but considerably less investigated task is to automatically tune the various parameters that control the search in game-playing programs. These parameters influence, for example, how aggressive the search algorithm is in pruning unpromising parts of the search tree and which lines are explored in more depth. The key problem here is that these parameters are intertwined with the search algorithm and cannot be optimized independently, making the process very tedious and expensive.

There have been a few attempts to use ▸ explanation-based learning to automatically learn predicates that indicate which branches of the search tree are the most promising to follow. These approaches are quite related to various uses of ▸ explanation-based learning in planning, but these could not be successfully be carried over to game-tree search.

Björnsson and Marsland (2003) present a *gradient descent* approach that minimizes the total number of game positions that need to be searched in order to successfully solve a number of training problems. The idea is to adjust each parameter in proportion to its sensitivity to changes in the number of searched nodes, which is estimated with additional searches. The amount of positions that can be searched for each training position is bounded in order to avoid infinite solution times for individual problems, and simulated annealing is used to ensure convergence.

Monte Carlo Tree Search

Automated tuning of evaluation functions and search control parameters does not work well for all games. For many years, research in computer Go has not made much progress with conventional search-based and pattern learning algorithms. However, a breakthrough came when Monte Carlo techniques could be combined with tree search algorithms. The basic algorithm interleaves four phases (Browne et al. 2012):

1. *Selection:* select a node of the current search tree for expansion
2. *Expansion:* generate one (or more) of the successor nodes for the selected node
3. *Simulation:* starting from these nodes, simulate a game until a terminal state is reached
4. *Backpropagation:* propagate the observed result back to the root of the tree

The best known of such methods, UCT, may be viewed as the extension of the UCB method for solving ▸ k-armed bandit problems to search trees (Kocsis and Szepesvári 2006). In the selection phase, UCT computes the following term for choosing the next node at each interior node of the current tree:

$$\mathrm{UCT} = \bar{X}_j + C \cdot \sqrt{\frac{2 \ln n}{n_j}} \qquad (1)$$

where X_j is the average reward that has been observed at node j, n_j is the number of times the node has been visited, and n is the number of times its predecessor has been visited. Clearly, it can be seen that nodes with a high utility are generally preferred (exploitation), but the second term also increases the chances that nodes that have been rarely visited are selected (exploration). The parameter C can be adjusted to trade off exploration and exploitation. From the selected node, a single random rollout is conducted, and its outcome is used to adapt the X_j values in all visited nodes in the search tree.

MCTS is generally applicable but has been particularly successful in game playing, most notably in Computer Go. In particular, AlphaGo (Silver et al. 2016), which employs deep learning for training value networks to evaluate positions and policy networks to bias the simulation phase of MCTS towards promising moves, became the first computer player to beat a world-class Go player in a celebrated 5-game match in March 2016.

Opening Book Learning

Human game players not only rely on their ability to estimate the value of moves and positions but are often also able to play certain positions "by heart," i.e., without having to think about their next move. This is the result of home preparation, opening study, and *rote learning* of important lines and variations. As computers do not forget, the use of an opening book provides an easy way for increasing their playing strength. However, the construction of such opening books can be quite laborious, and the task of keeping it up-to-date is even more challenging.

Commercial game-playing programs, in particular chess programs, have thus resorted to tools that support the automatic construction of opening from large game databases. The key challenge here is that one cannot rely on statistical information alone: a move that has been successfully employed in hundreds of games may be refuted in a single game. Donninger and Lorenz (2006) describe an approach that evaluates the "goodness" of a move based on a heuristic formula that has been found by experimentation. This value is then added to the result of a regular alpha-beta search. The technique has been so successful that the chess program HYDRA, probably the strongest chess program today, has abandoned conventionally large man-made (and therefore error-prone) error books. Similar techniques have also been used in games like Othello (Buro 2002).

Pattern Discovery

In addition to databases of common openings and huge game collections, which are mostly used for the tuning of evaluation functions or the automatic generation of opening books (see above), many games or subgames have already been solved, i.e., databases are available in which the game-theoretic value of positions of these subgames can be looked up. For example, in chess all endgames with up to six pieces and in checkers all ten-piece endgames have been solved (Schaeffer et al. 2003). Other games, such as Connect-4, are solved completely, i.e., all possible positions have been evaluated, and the game-theoretic value of the starting position has been determined. Many of these databases are readily available; some of them (in the domains of chess, Connect-4, and tic-tac-toe) are part of the UCI repository for machine learning databases.

The simplest learning task is to train a classifier that is able to decide whether a given game position is a game-theoretical win or loss (or draw). In many cases, this is insufficient. For example, in the chess endgame king-rook-king, any position in which the white rook cannot be immediately captured and in which black is not a stalemate is, in principle, won by white. However, in order to actually win the game, it is not sufficient to simply make moves that avoid rook captures and stalemates. Thus, most databases contain the maximal number of moves that are needed for winning the position. Predicting this is a much harder, largely unsolved problem (some recent work can be found in Sadikov and Bratko 2006). In addition to the game-specific knowledge that could be gained by the extraction of patterns that are indicative of won positions, another major application could be a knowledge-based compression of these databases (the collection of all perfect-play chess endgame databases with up to six men is 1.2 TB in a very compressed database format; the win/loss checkers databases with up to ten men contain about 4×10^{13} positions compressed into 215 GB Schaeffer et al. 2003).

Player Modeling

Player modeling is an important research area in game playing, which can serve several purposes. The goal of *opponent modeling* is to improve the capabilities of the machine player by allowing it to adapt to its opponent and exploit his weaknesses. Even if a game-theoretical optimal solution to a game is known, a system that has the capability to model its opponent's behavior may obtain a higher reward. Consider, for example, the game of *rock-paper-scissors* aka *RoShamBo*, where either player can expect to win one third of the games (with one third of draws) if both players play their optimal strategies (i.e., randomly select one of their three moves). However, against a player that always plays *rock*, a player that is able to adapt his strategy to always playing *paper* can maximize his reward, while a player that sticks with the "optimal" random strategy will still win only one third of the games. One of the grand challenges in this line of work is a game like poker, where opponent modeling is crucial to improve over game-theoretical optimal play (Billings et al. 2002).

Player modeling is also of increasing importance in commercial computer games (see below). For one, ▸ behavioral cloning techniques could be used to increase the playing strength or credibility of artificial characters by copying the strategies of expert human players. Moreover, the playing strength of the characters can be adapted to the increasing skill level of the human player. Finally, agents that can be trained by nonprogrammers can also play an important role. For example, in massive multiplayer online role-playing games (MMORPGs), an avatar that is trained to simulate a user's game-playing behavior could take his creator's place at times when the human player cannot attend to his game character.

Commercial Computer Games

In recent years, the computer game industry has discovered artificial intelligence as a necessary ingredient to make games more entertaining and challenging, and, vice versa, AI has discovered computer games as an interesting and rewarding application area (Laird and van Lent 2001). In comparison to conventional strategy games, computer game applications are more demanding, as the agents in these games typically have to interact with a large number of partner or enemy agents in a highly dynamic, real-time environment, with incomplete knowledge about its states. Tasks include off-line or online player modeling (see above), virtual agents with learning capabilities, optimization of plans and processes, etc.

Computer players in games are often controlled with scripts. *Dynamic scripting* (Spronck et al. 2006) is an online ▸ reinforcement learning technique that is designed to be integrated into scripting languages of game-playing agents. Contrary to conventional reinforcement learning agents, it updates the weights of all actions for a given state simultaneously. This sacrifices guaranteed convergence, but this is desirable in a highly dynamic game environment. The approach was successfully applied to improving the strength of computer-controlled characters and increasing the entertainment value of the game by automated scaling of the difficult level of the game AI to the human player's skill level. Similar to the problem of constructing suitable features for the use in evaluation functions, the basic tactics of the computer player had to be handcoded. Ponsen et al. (2006) extend dynamic scripting with an ▸ evolutionary algorithm for automatically constructing the tactical behaviors.

Machine learning techniques are not only used for controlling players but also for tasks like skill estimation, for example, TrueSkill^{TM} (Herbrich et al. 2007), a Bayesian skill rating system which is used for ranking players in games on the Microsoft's Xbox 360. SAGA-ML (Southey et al. 2005) is a machine learning system for supporting game designers in improving the playability of a game.

Despite the large commercial potential, research in this area has just started, and the number of workshops and publications on this topic is rapidly increasing. For a list of commercial games using AI techniques, we refer to http://www.gameai.com.

Cross-References

▸ Samuel's Checkers Player
▸ TD-Gammon

Recommended Reading

Amit A, Markovitch S (2006) Learning to bid in bridge. Mach Learn 63(3):287–327.

Baxter J, Tridgell A, Weaver L (2000) Learning to play chess using temporal differences. Mach Learn 40(3):243–263.

Billings D, Peña L, Schaeffer J, Szafron D (2002) The challenge of poker. Artif Intell 134(1–2):201–240. Special Issue on Games, Computers and Artificial Intelligence

Björnsson Y, Marsland TA (2003) Learning extension parameters in game-tree search. Inf Sci 154(3–4):95–118.

Bowling M, Fürnkranz J, Graepel T, Musick R (2006) Special issue on machine learning and games. Mach Learn 63(3).

Browne CB, Powley E, Whitehouse D, Lucas SM, Cowling PI, Rohlfshagen P, Tavener S, Perez D, Samothrakis S, Colton S (2012) A survey of Monte Carlo tree search methods. IEEE Trans Comput Intell AI Games 4(1):1–43

Buro M (2002) Improving heuristic mini-max search by supervised learning. Artif Intell 134(1–2):85–99. Special Issue on Games, Computers and Artificial Intelligence

Donninger C, Lorenz U (2006) Innovative opening-book handling. In: van den Herik HJ, Shun-Chin Hsu, Donkers HHLM (eds) Advances in computer games, vol 11. Springer, Berlin/New York

Fürnkranz J (2001) Machine learning in games: a survey. In: Fürnkranz J, Kubat M (eds) Machines that learn to play games, chapter 2. Nova Science Publishers, Huntington, pp 11–59.

Fürnkranz J, Kubat M (eds) (2001) Machines that learn to play games. Volume 8 of advances in computation: theory and practice. Nova Science Publishers, Huntington.

Ghory I (2004) Reinforcement learning in board games. Technical report CSTR-04-004, Department of Computer Science, University of Bristol, Bristol.

Herbrich R, Minka T, Graepel T (2007) Trueskill^{tm}: a Bayesian skill rating system. In: Schölkopf B, Platt JC, Hoffman T (eds) Advances in neural information processing systems (NIPS-06), Vancouver, vol 19. MIT Press, pp 569–576

Kocsis L, Szepesvári C (2006) Bandit based monte-carlo planning. In: Proceedings of the 17th European conference on machine learning, ECML'06. Springer, Berlin/Heidelberg, pp 282–293

Laird JE, van Lent M (2001) Human-level AI's Killler application: interactive computer games. AI Mag 22(2):15–26

Michie D (1963) Experiments on the mechanization of game-learning – Part I. Characterization of the model and its parameters. Comput J 6:232–236

Ponsen M, Muñoz-Avila H, Spronck P, Aha DW (2006) Automatically generating game tactics via evolutionary learning. AI Mag 27(3):75–84.

Sadikov A, Bratko I (2006) Learning long-term chess strategies from databases. Mach Learn 63(3): 329–340

Schaeffer J, van den Herik HJ (eds) (2002) Chips challenging champions: games, computers and artificial intelligence. North-Holland Publishing, Amsterdam. Reprint of a Special Issue of Artificial Intelligence 134(1–2)

Schaeffer J, Björnsson Y, Burch N, Lake R, Lu P, Sutphen S (2003) Building the checkers 10-piece endgame databases. In: van den Herik HJ, Iida H, Heinz EA (eds) Advances in computer games, vol 10. Springer, Graz, pp 193–210.

Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever I, Lillicrap T, Leach M, Kavukcuoglu K, Graepel T, Hassabis D (2016) Mastering the game of Go with deep neural networks and tree search. Nature. 529: 484–489.

Southey F, Xiao G, Holte RC, Trommelen M, Buchanan JW (2005) Semi-automated gameplay analysis by machine learning. In: Young RM, Laird JE (eds) Proceedings of the 1st artificial intelligence and interactive digital entertainment conference (AIIDE-05). AAAI Press, Marina del Rey, pp 123–128

Spronck P, Ponsen MJV, Sprinkhuizen-Kuyper IG, Postma EO (2006) Adaptive game AI with dynamic scripting. Mach Learn 63(3):217–248.

Machine Learning for IT Security

Philip K. Chan
Florida Institute of Technology, Melbourne, FL, USA

Definition

The prevalence of information technology (IT) across all segments of society, greatly improves the accessibility of information, however, it also provides more opportunities for individuals to act with malicious intent. Intrusion detection is the task of identifying attacks against computer systems and networks. Based on data/behavior observed in the past, machine learning methods can automate the process of building detectors for identifying malicious activities.

Motivation and Background

Cyber security often focuses on preventing attacks using authentication, filtering, and encryption techniques, but another important facet is detecting attacks once the preventive measures are breached. Consider a bank vault: thick steel doors prevent intrusions, while motion and heat sensors detect intrusions. Prevention and detection complement each other to provide a more secure environment.

How do we know if an attack has occurred or has been attempted? This requires analyzing huge volumes of data gathered from the network, host, or file systems to find suspicious activities. Two general approaches exist for this problem: *misuse detection* (also known as *signature detection*), where we look for patterns signaling well-known attacks, and ▸ *anomaly detection*, where we look for deviations from normal behavior.

Misuse detection usually works reliably on known attacks (though false alarms and missed detections are not uncommon), but has the obvious disadvantage of not being capable of detecting new attacks. Though anomaly detection can detect novel attacks, it has the drawback of not being capable of discerning intent; it can only signal that some event is unusual, but not necessarily hostile, thus generating false alarms. A desirable system would employ both approaches. Misuse detection methods are more well understood and widely applied; however, anomaly detection is much less understood and more challenging.

Can we automate the process of building software for misuse and anomaly detection? Machine learning techniques hold promise in efficiently analyzing large amounts of recent activities, identifying patterns, and building detectors.

Besides computer attacks, spam email messages, though not intended to damage computer systems or data, are annoying and waste system resources. To construct spam detectors from large amounts of email messages, machine learning techniques have been used (see "References" and "Recommended Reading" for more).

Structure of Learning System

Machine learning can be used to construct models for misuse as well as anomaly detection.

Misuse Detection

For misuse detection, the machine learning goal is to identify characteristics of known attacks. One approach is to learn the difference between attacks and normal events, which can be casted as a classification problem. Given examples of labeled attacks and normal events, a learning algorithm constructs a model that differentiates attacks from normal events.

Lee et al. (1999) apply machine learning to detect attacks in computer networks. They first identify frequent episodes, associations of features that frequently appear within a time frame, in attack and normal data separately. Frequent episodes that only appear in attack data help construct features for the models. For example, if the SYN flag is set for a http connection is a frequent episode within 2 s and the episode only appears in the attack data, a feature is constructed for the number of http connections with the SYN flag set within a period of 2 s. Using RIPPER and based on different sets of features, they construct three models: traffic, host-based traffic, and content models. The three models are then combined using meta-learning.

Ghosh and Schwartzbard (1999) use neural networks to identify attacks in operating systems. Based on system calls in the execution traces of normal and attack programs, they first identify a number of "examplar" sequences of system calls. For each system call sequence, they calculate the distance from the examplar sequences. The number of input nodes for the neural network is equal to the number of examplars and values for the input nodes are distances from those examplar sequences. The value for the output node is whether the system call sequence is from an attack or normal program.

Anomaly Detection

For anomaly detection, the machine learning goal is to characterize normal behavior. The learned models of normal behavior are then used to identify events that are anomalies, events that deviate from the models. Since anomalies are not always attacks, to reduce false alarms, the learned models usually provide a scoring mechanism to indicate the degree of anomaly.

Warrender et al. (1999) identify anomalies in system calls in the operating systems. The model is a table of system call sequences from execution traces of normal programs. During detection, a sequence that is not in the table or occurs less than 0.001 % in the training data is considered a mismatch. The number of mismatches within a locality frame of 20 sequences is the anomaly score.

Mahoney and Chan (2003) introduce the LERAD algorithm for learning rules that identify anomalies in network traffic. LERAD first uses a randomized algorithm to generate candidate rules that represent associations. It then finds a set of high quality rules that can succinctly cover the training data. Each rule has an associated probability of violating the rule. During detection, based on the probability, LERAD provides a score for anomalous events that do not conform to the rules in the learned model.

Misuse Detection: Schultz et al. (2001) with program executables, Maxion and Townsend (2002) with user commands.

Anomaly Detection: Sekar et al. (2001) with program execution, Apap et al. (2002) with Windows Registry, Anderson et al. (1995) with system resources, Lane and Brodley (1999) with user commands.

Spam detection: Bratko et al. (2006) with text, Fumera et al. (2006) with text and embedded images.

Cross-References

- ▸ Anomaly Detection
- ▸ Association Rule
- ▸ Classification

Recommended Reading

Anderson D, Lunt T, Javitz H, Tamaru A, Valdes A (1995) Detecting unusual program behavior using the statistical component of the next-generation intrusion detection expert system (NIDES). Technical report SRI-CSL-95-06, SRI

Apap F, Honig A, Hershkop S, Eskin E, Stolfo S (2002) Detecting malicious software by monitoring anomalous windows registry accesses. In: Proceeding of fifth international symposium on recent advances in intrusion detection (RAID), Zurich, pp 16–18

Bratko A, Filipic B, Cormack G, Lynam T, Zupan B (2006) Spam filtering using statistical data compression models. J Mach Learn Res 7:2673–2698

Fumera G, Pillai I, Roli F (2006) Spam filtering based on the analysis of text information embedded into images. J Mach Learn Res 7:2699–2720

Ghosh A, Schwartzbard A (1999) A study in using neural networks for anomaly and misuse detection. In: Proceeding of 8th USENIX security symposium, Washington, DC, pp 141–151

Lane T, Brodley C (1999) Temporal sequence learning and data reduction for anomaly detection. ACM Trans Inf Syst Secur 2(3):295–331

Lee W, Stolfo S, Mok K (1999) A data mining framework for building intrusion detection models. In: IEEE symposium on security and privacy, pp 120–132

Mahoney M, Chan P (2003) Learning rules for anomaly detection of hostile network traffic. In: Proceeding of IEEE international conference data mining, Melbourne, pp 601–604

Maxion R, Townsend T (2002) Masquerade detection using truncated command lines. In: Proceeding of international conference dependable systems and networks (DSN), Washington, DC, pp 219–228

Schultz M, Eskin E, Zadok E, Stolfo S (2001) Data mining methods for detection of new malicious executables. In: Proceeding of IEEE symposium security and privacy, Oakland, pp 38–49

Sekar R, Bendre M, Dhurjati D, Bollinen P (2001) A fast automaton-based method for detecting anomalous program behaviors. In: Proceeding of IEEE symposium security and privacy, Oakland, pp 144–155

Warrender C, Forrest S, Pearlmutter B (1999) Detecting intrusions using system calls: alternative data models. In: IEEE symposium on security and privacy, Los Alamitos, pp 133–145

Manhattan Distance

Susan Craw
Robert Gordon University, Aberdeen, UK

Synonyms

City block distance; L_1-distance; 1-norm distance; Taxicab norm distance

Definition

The Manhattan distance between two points $\mathbf{x} = (x_1, x_2, \ldots x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots y_n)$ in n-dimensional space is the sum of the distances in each dimension:

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \mid x_i - y_i \mid$$

It is called the Manhattan distance because it is the distance a car would drive in a city (e.g., Manhattan) where the buildings are laid out in square blocks and the straight streets intersect at right angles. This explains the other terms city block and taxicab distances. The terms L_1 and 1-norm distances are the mathematical descriptions of this distance.

Cross-References

- ► Case-Based Reasoning
- ► Nearest Neighbor

Margin

Definition

In a ► Support Vector Machine, a *margin* is the distance between a hyperplane and the closest example.

Cross-References

- ► Support Vector Machines

Market Basket Analysis

- ► Basket Analysis

Markov Chain

- ► Markov Process

Markov Chain Monte Carlo

Claude Sammut
The University of New South Wales, Sydney,
NSW, Australia

Synonyms

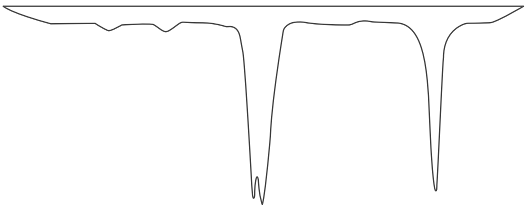
MCMC

Definition

A Markov Chain Monte Carlo (MCMC) algorithm is a method for sequential sampling in which each new sample is drawn from the neighborhood of its predecessor. This sequence forms a ► Markov chain, since the transition probabilities between sample values are only dependent on the last sample value. MCMC algorithms are well suited to sampling in high-dimensional spaces.

Motivation

Sampling from a probability density function is necessary in many kinds of approximation, including Bayesian inference and other applications in Machine Learning. However, sampling is not always easy, especially in high-dimensional spaces. Mackay (2003) gives a simple example to illustrate the problem. Suppose we want to find the average concentration of plankton in a lake, whose profile looks like this:



If we do not know the depth profile of the lake, how would we know where to sample from? If we take a boat out, would we have to sample almost exhaustively by fixing a grid on the surface of the lake and sinking our instrument progressively deeper, sampling at fixed intervals until we hit the bottom? This would be prohibitively expensive and if we had a similar problem, but with more dimensions, the problem becomes intractable. If we try to simplify the problem by drawing a random sample, how do we ensure that enough samples are taken from the canyons in the lake and not just the shallows, which account for most of the surface area?

The Algorithm

The general approach adopted in MCMC algorithms is as follows. We start sampling in some random initial state, represented by vector, $\boldsymbol{x}$. At each state, we can evaluate the probability density function, $P(\boldsymbol{x})$. We then choose a candidate next state, $\boldsymbol{x}'$, near the current state and evaluate $P(\boldsymbol{x}')$. Comparing the two, we decide whether to accept or reject the candidate. If we accept it, the candidate becomes the new current state and the process repeats for a fixed number of steps or until some convergence criterion is satisfied.

The Metropolis Algorithm

There are several variants of the general algorithm presented above. Each variant must specify how a candidate state is proposed and what criterion should be used to accept or reject the candidate. The Metropolis algorithm assumes that the next candidate is drawn from a symmetric distribution, $Q(x)$, centered on the current state, for example, a Gaussian distribution (Metropolis et al. 1953; Metropolis and Ulam 1949). This distribution is called the *proposal distribution.* The Metropolis algorithm is shown in Algorithm 1.

To decide if a candidate should be accepted or rejected, the algorithm calculates,

$$\alpha = \frac{P(x')}{P(x_i)}$$

where $\boldsymbol{x}_i$ is the current state and $\boldsymbol{x}'$ is the candidate state. If $\alpha > 1$, the candidate is immediately accepted. If $\alpha < 1$, then a stochastic choice is made with the candidate being accepted with probability α, otherwise, it is rejected.

Algorithm 1 The Metropolis Algorithm

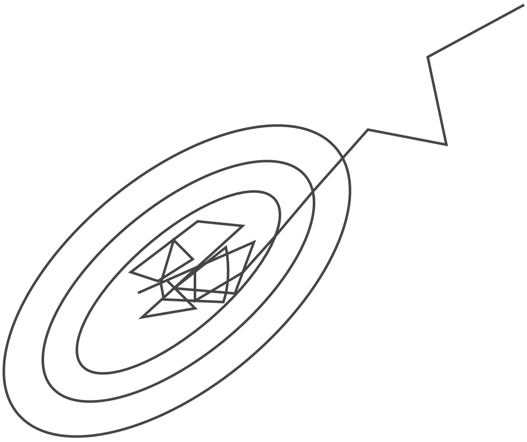
```
Given: target probability density function $P(\boldsymbol{x})$
    a proposal distribution, $Q$, e.g., a Gaussian
    the number of iterations, $N$
Output: a set of samples $\{x_i\}$ drawn from $P(\boldsymbol{x})$
Randomly select initial state vector, $x_0$
for $i = 0$ to $N - 1$
  create a new candidate $\boldsymbol{x}' = \boldsymbol{x}_i + \Delta\boldsymbol{x}$,
    where $\Delta\boldsymbol{x}$ is randomly chosen from $Q(\Delta\boldsymbol{x})$
  set $\alpha = \frac{P(x')}{P(x_i)}$
  if $\alpha \geq 1$ or with probability $\alpha$
    accept the new candidate and set $\boldsymbol{x}_{i+1} = \boldsymbol{x}'$
  else
    reject the candidate and set $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i$
```

Hastings (1970) introduced a variant, the Metropolis–Hastings algorithm, which allows the proposal distribution to be asymmetric. In this case, the accept/reject calculation is:

$$\alpha = \frac{P(x')Q(x_i; x')}{P(x_i)Q(x'; x_i)}$$

Burn-In and Convergence

It can be difficult to decide how many iterations are needed before an MCMC algorithm achieves a stable distribution. Several factors affect the length of the Markov chain needed. Depending on the start state, many of the initial samples may have to be discarded, called *burn-in*, as illustrated below. The ellipses represent contours of the distribution.



The variance of the proposal distribution can also affect the chain length. If the variance is large, the jumps are large, meaning that there is varied sampling. However, this is also likely to mean that fewer samples are accepted. Narrowing the variance should increase acceptance but may require a long chain to ensure wide sampling, which is particularly necessary if the distribution has several peaks. See Andrieu et al. (2003) for a discussion of methods for improving convergence times.

Gibbs Sampling

An application of MCMC is inference in a ▶ Bayesian network, also known as ▶ Graphical Models. Here, we sample from evidence variables to find a probability for non-evidence variables. That is, we want to know what unknowns we can derive from the knowns and with what probability. Combining the evidence across a large network is intractable because we have to take into account all possible interactions of all variables, subject to the dependencies expressed in the network. Since there are too many combinations to compute in a large network, we approximate the solution by sampling. The Gibbs sampler is a special case of the Metropolis–Hastings algorithm that is well suited to sampling from distributions over two or more dimensions. It proceeds as in Algorithm 1, except that when a new candidate is generated, only one dimension is allowed to change while all the others are held constant. Suppose we have n dimensions and $\boldsymbol{x} = (x_1, \ldots, x_n)$. One complete pass consists of jumping in one dimension, conditioned on the values for all the other dimensions, then jumping in the next dimension, and so on. That is, we initialise x to some value, and then for each xi we resample $P(x_i|x_{j=6i})$ for j in $1 \ldots n$. The resulting candidate is immediately accepted. We then iterate, as in the usual Metropolis algorithm.

Cross-References

▶ Bayesian Network
▶ Graphical Models
▶ Learning Graphical Models
▶ Markov Chain

Recommended Reading

MCMC is well covered in several text books. Mackay (2003) gives a thorough and readable introduction to MCMC and Gibbs Sampling. Russell and Norvig (2009) explain MCMC in the context of approximate inference for Bayesian networks. Hastie et al. (2009) also give a more technical account of sampling from the posterior. Andrieu et al. (2003) Machine Learning paper gives a thorough introduction to MCMC for Machine Learning. There are also some excellent tutorials on the web including Walsh (2004) and Iain Murray's video tutorial (Murray 2009) for machine learning summer school.

Andrieu C, DeFreitas N, Doucet A, Jordan MI (2003) An introduction to MCMC for machine learning. Mach Learn 50(1):5–43

Hastie T, Tibshirani R, Friedman J (2009) The elements of statistical learning: data mining, inference and perception, 2nd edn. Springer, New York

Hastings WK (1970) Monte Carlo sampling methods using Markov chains and their applications. Biometrika 57:97–109

Mackay DJC (2003) Information theory, inference and learning algorithms. Cambridge University Press, Cambridge

Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller A, Teller H (1953) Equations of state calculations by fast computing machines. J Chem Phys 21:1087–1091

Metropolis N, Ulam S (1949) The Monte Carlo method. J Am Stat Assoc 44(247):335–341

Murray I (2009) Markov chain Monte Carlo. http://videolectures.net/mlss09uk_murray_mcmc/. Retrieved 25 July 2010

Russell S, Norvig P (2009) Artificial intelligence: a modern approach, 3rd edn. Prentice Hall, Englewood Cliffs

Walsh B (2004) Markov chain Monte Carlo and Gibbs sampling. http://nitro.biosci.arizona.edu/courses/EEB581-2004/handouts/Gibbs. Retrieved 25 July 2010

Markov Decision Processes

William Uther
NICTA and The University of New South Wales,
Sydney, NSW, Australia

Synonyms

Policy search

Definition

A *Markov Decision Process* (MDP) is a discrete, stochastic, and generally finite model of a system to which some external control can be applied. Originally developed in the Operations Research and Statistics communities, MDPs, and their extension to ▸ Partially Observable Markov Decision Processes (POMDPs), are now commonly used in the study of ▸ reinforcement learning in the Artificial Intelligence and Robotics communities (Bellman 1957; Bertsekas and Tsitsiklis 1996; Howard 1960; Puterman 1994). When used for reinforcement learning, firstly the parameters of an MDP are learned from data, and then the MDP is processed to choose a behavior.

Formally, an MDP is defined as a tuple: $< \mathcal{S}, \mathcal{A}, T, R >$, where $\mathcal{S}$ is a discrete set of states, $\mathcal{A}$ is a discrete set of actions, $T : \mathcal{S} \times \mathcal{A} \rightarrow (\mathcal{S} \rightarrow R)$ is a stochastic transition function, and $R : \mathcal{S} \times \mathcal{A} \rightarrow R$ specifies the expected reward received for performing the given action in each state.

An MDP carries the *Markov* label because both the transition function, T, and the reward function, R, are Markovian; i.e., they are dependent only upon the current state and action, not previous states and actions. To be a valid transition function, the distribution over the resulting states, $\mathcal{S} \rightarrow R$, must be a valid probability distribution, i.e., non-negative and totalling 1. Furthermore, the expected rewards must be finite.

The usual reason for specifying an MDP is to find the optimal set of actions, or *policy*, to perform. We formalize the optimality criteria below. Let us first consider how to represent a policy. In its most general form the action, $a \in \mathcal{A}$, indicated by a policy, π, might depend upon the entire history of the agent; $\pi : (\mathcal{S} \times \mathcal{A})^* \times \mathcal{S} \rightarrow \mathcal{A}$. However, for each of the common optimality criteria considered below a Markov policy, $\mathcal{S} \rightarrow \mathcal{A}$, will be sufficient. i.e., for every MDP, for each of the optimality criteria below, there exists a Markov policy that performs as well as the best full policy. Similarly, there is no requirement for an MDP that a policy be stochastic or mixed.

Optimality Criteria

Informally, one wants to choose a policy so as to maximise the long term sum of immediate rewards. Unfortunately the naive sum, $\sum_{t=0}^{\infty} r_t$ where r_t is the expected immediate reward received at time t, usually diverges. There are different optimality criteria that can than be used as alternatives.

Finite Horizon

The easiest way to make sure that the sum of future expected rewards is bounded is to only consider a fixed, finite time into the future; i.e., find a policy that maximises $\sum_{t=0}^{n} r_t$ for each state.

Infinite Horizon Discounted

Rather than limiting the distance we look into the future, another approach is to *discount* rewards we will receive in the future by a multiplicative factor, γ, for each time-step. This can be justified as an inflation rate, as an otherwise unmodelled probability that the simulation ends each time-step, or simply as a mathematical trick to make the criteria converge. Formally we want a policy that maximises $\sum_{t=0}^{\infty} \gamma^t r_t$ for each state.

Average Reward

Unfortunately, the infinite horizon discounted optimality criterion adds another parameter to our model: the discount factor. Another approach is to optimize the average reward per time-step, or *gain*, by finding a policy that maximizes $\lim_{n\rightarrow\infty} \frac{1}{n} \sum_{t=0}^{n} r_t$ for each state. This is very similar to using *sensitive discount optimality*; finding a policy that maximizes the infinite horizon discounted reward as the discount factor approaches 1, $\lim_{\gamma\rightarrow 1} \sum_{t=0}^{\infty} \gamma^t r_t$, for each state.

When maximizing average reward, any finite deviation from the optimal policy will have negligible effect on the average over an infinite timeframe. This can make the agent "lazy." To counteract this, often a series of increasingly strict optimality criteria are used. The first is the "gain" optimality criterion given above – optimizing the long term average reward. The next is a "bias" optimality which selects from among all gain optimal policies the ones that also optimize transient initial rewards.

Value Determination

For the finite horizon, infinite horizon discounted, or bias optimality criteria, the optimality criteria can be calculated for each state, or for each state-action pair, giving a *value function*. Once found, the value function can then be used to find an optimal policy.

Bellman Equations

The standard approach to finding the value function for a policy over an MDP is a dynamic programming approach using a recursive formulation of the optimality criteria. That recursive formulation is known as the Bellman equation.

There are two, closely related, common forms for a value function; the state value function, $V : \mathcal{S} \to R$ and the state-action value function, $Q : \mathcal{S} \times \mathcal{A} \to R$. For a finite horizon undiscounted optimality criterion with time horizon n and policy π:

$$\begin{aligned}
\mathrm{Q}_n^\pi(s,a) &= \sum_{t=0}^{n} r_t \\
&= R(s,a) + \mathrm{E}_{s' \in T(s,a)} V_{n-1}^\pi(s') \\
&= R(s,a) + \sum_{s' \in \mathcal{S}} T(s,a)(s') V_{n-1}^\pi(s') \\
V_n^\pi(s) &= \mathrm{Q}_n^\pi(s, \pi(s))
\end{aligned}$$

For the infinite horizon discounted case:

$$\begin{aligned}
\mathrm{Q}^\pi(s,a) &= R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a)(s') V^\pi(s') \\
V^\pi(s) &= \mathrm{Q}^\pi(s, \pi(s))
\end{aligned}$$

These equations can be turned into a method for finding the value function by replacing the equality with an assignment:

$$\begin{aligned}
\mathrm{Q}^\pi(s,a) &\leftarrow R(s,a) \\
&\quad + \gamma \sum_{s' \in \mathcal{S}} T(s,a)(s') \mathrm{Q}^\pi(s', \pi(s'))
\end{aligned}$$

As long as this update rule is followed infinitely often for each state/action pair, the Q-function will converge.

Prioritised sweeping: Rather than blindly updating each state/action, intelligent choice of where to update will significantly speed convergence. One technique for this is called Prioritized Sweeping (Moore 1993; Andre et al. 1997).

A priority queue of states is kept. Initially one complete pass of updates over all states is performed, but thereafter states are updated in the order they are pulled from the priority queue. Any time the value of a state, $V^\pi(s)$, changes, the priorities of all states, s', that can reach state s are updated; we update $\{s' | T(s', \pi(s'))(s) \neq 0\}$. The priorities are increased by the absolute change in $V^\pi(s)$.

The effect of the priority queue is to focus computation where values are changing rapidly.

Linear Programming Solutions

Rather than using the Bellman equation and dynamic programming, an alternative approach is to set up a collection of inequalities and use linear programming to find an optimal value function. In particular if we minimize,

$$\sum_{s \in \mathcal{S}} V^\pi(s)$$

subject to the constraints

$$\begin{aligned}
\forall_{s \in \mathcal{S}} 0 \leq V^\pi(s) - [R(s,a) \\
+ \gamma \sum_{s' \in \mathcal{S}} T(s,a)(s') V^\pi(s')],
\end{aligned}$$

then the resulting V^π accurately estimates the expected sum of discounted reward.

Bellman Error Minimization

A third approach to value determination is similar to the dynamic programming solution above. Rather than replacing the equality in the Bellman equation with an assignment, it turns the equation into an error function and adjusts the Q function to minimise the sum of squared Bellman residuals

(Baird 1995):

$$\begin{aligned}\text{Residual}(s) &= \text{Q}^{\pi}(s,\ a) - [R(s,\ a) \\ &\quad + \gamma \sum_{s' \in \mathcal{S}} T(s,\ a)(s')\text{Q}^{\pi}(s',\ \pi(s'))]\ \text{Err} \\ &= \sum_{s \in \mathcal{S}} \text{Residual}(s)^2\end{aligned}$$

Control Methods

The previous section gave us a way to obtain a value function for a particular policy, but what we usually need is a good policy, not a value function for the policy we already have. For an optimal policy, for each state:

$$\pi(s) = argmax_{a \in \mathcal{A}} Q^{\pi}(s, a)$$

If a policy, π, is not optimal then its value function can be used to find a better policy, π'. It is common to use the greedy policy for the value function:

$$\pi'(s) \leftarrow argmax_{a \in \mathcal{A}} Q^{\pi}(s, a)$$

This process can be used iteratively to find the optimal policy.

Policy iteration: Policy iteration alternates between value determination and greedy policy updating steps until convergence is achieved. The algorithm starts with a policy, π_1. The value function is calculated for that policy, $V^{\pi 1}$. A new policy is then found from that value function, π_2. This alternation between finding the optimal value function for a given policy and then improving the policy continues until convergence. At convergence the policy is optimal.

Value iteration: Rather than explicitly updating the policy, value iteration works directly with the value function. We define an update,

$$\begin{aligned}\text{Q}(s, a) &\leftarrow R(s, a) \\ &\quad + \gamma \sum_{s' \in \mathcal{S}} T(s, a)(s') \max_{a' \in A} \text{Q}(s', a'),\end{aligned}$$

with a maximization step included. As long as this update is performed often enough in each state, Q will converge. Once Q has converged, the greedy policy will be optimal.

Mixed policy iteration: The two previous methods, policy and value iteration, are two extremes of a spectrum. In practice updates to the policy and value function can occur asynchronously as long as the value and policy in each state are updated often enough.

Representations

In the above discussion we have discussed a number of functions, but not discussed how these functions are represented. The default representation is an array or tabular form which has no constraints on the function it can represent. However, the ▸ curse of dimensionality suggests that the number of states will, in general, be exponential in the problem size. This can make even a single complete iteration over the state space intractable. One solution is to represent the functions in a more compact form so that they can be updated efficiently. This approach is known as *function approximation*. Here we review some common techniques.

A class of representations is chosen to represent the functions we need to process: e.g., the transition, T, reward, R, Value, V or Q, and/or policy, π, functions. A particular function is selected from the chosen class by a parameter vector, $\boldsymbol{\theta}$.

There are two important questions that must be answered by any scheme using function approximation; does the resulting algorithm converge to a solution, and does the resulting solution bear any useful relationship with the optimal solution?

A simple approach when using a differentiable function to represent the value function is to use a form of ▸ temporal difference learning. For a given state, s, and action, a, the Bellman equation is used to calculate a new value, $Q^{new}(s, a)$, and then $\boldsymbol{\theta}$ is updated to move the value function toward this new value. This gradient based approach usually has a learning rate, $\alpha \in [0, 1]$, to adjust the speed of learning.

$$\text{Q}^{\text{new}}(s, a) \leftarrow R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a)(s') V^{\text{old}}(s')$$

$$\Delta_{s,a}\theta = \alpha \frac{\partial \mathrm{Q}}{\partial \theta}(\mathrm{Q}^{\mathrm{new}}(s,a) - \mathrm{Q}^{01\mathrm{d}}(s,a))$$

This approach is known not to converge in general, although it does converge in some special cases. A similar approach with full Bellman Error minimization will not oscillate, but it may cause the $\boldsymbol{\theta}$ to diverge even as the Bellman residual converges.

Contraction mappings: The first class of function approximators that was shown to converge with the above update, apart from a complete tabular representation, was the class of contraction mappings (Gordon 1995). Simply put, these are function approximation classes where changing one value by a certain amount changes every other value in the approximator by no more than that amount. For example, linear interpolation and *tile coding* (Tile codings are also known as Cerebellar Motor Action Controllers (CMAC) in early work Albus 1981) are each contraction mappings whereas linear extrapolation is not.

Formally, let $\mathcal{S}$ be a vector space with max norm $||.||_\infty$. A function f is a contraction mapping if,

$$\forall a, b \in \mathcal{S}, \|f(a) - f(b)\|_\infty < \|a - b\|_\infty$$

The class of function approximations that form contraction mappings includes a number of common approximation techniques including *tile coding*. Tile coding represents a function as a linear combination of basis functions, $\boldsymbol{\phi}(s,a)$,

$$\hat{Q}(s,a) = \boldsymbol{\theta} \cdot \varphi(s,a),$$

where the individual elements of φ are binary features on the underlying state.

Linear approximations: The linear combination of basis functions can be extended beyond binary features. This will converge when temporal differencing updates are performed in trajectories through the state space following the policy being evaluated (Tsitsiklis and Van Roy 1997).

Variable resolution techniques: One technique for representing value functions over large state spaces is use a non-parametric representation. Munos gives a technique that introduces more basis functions for their approximation over time as needed (Munos and Moore 2001).

Dynamic Bayesian networks: Bayesian Networks are an efficient representation of a factored probability distribution. Dynamic Bayesian Networks use the Bayesian Network formalism to represent the transition function, $\mathcal{T}$, in an MDP (Guestrin et al. 2003). The reward and value functions are usually represented with linear approximations. The policy is usually represented implicitly by the value function.

Decision diagrams: Arithmetic Decision Diagrams (ADDs) are a compact way of representing functions from a factored discrete domain to a real range. ADDs can also be efficiently manipulated, with operators for the addition and multiplication of ADDs as well as taking the maximum of two ADDs. As the Bellman equation can be re-written using operators, it is possible to implement mixed policy iteration using this efficient representation St-Aubin et al. (2000).

Hierarchical representations: ▸ Hierarchical Reinforcement Learning factors out common substructure in the functions that represent an MDP in order to solve it efficiently. This has been done in many different ways. Dietterich's *MAXQ* hierarchy allowed a prespecified hierarchy to re-use common elements in a value function (Dietterich 2000). Sutton's *Options* framework focussed on temporal abstraction and re-use of policy elements (Sutton et al. 1998). Moore's *Airports* hierarchy allowed automatic decomposition of a problem where the specific goal could change over time, and so was made part of the state (Moore et al. 1999). Andre's *A-Lisp* system takes the hierarchical representation to an extreme by building in a Turing complete programming language (Andre and Russell 2002).

Greedy Algorithms Versus Search

In the previous sections the control problem was solved using a greedy policy for a value function. If the value function was approximate, then the resulting policy may be less than optimal. Another approach to improving the policy is to introduce search during execution. Given the current state, the agent conducts a forward search looking for the sequence of actions that produces the best intermediate reward and resulting state value combination.

These searches can be divided into two broad categories: deterministic and stochastic searches. Deterministic searches, such as LAO∗ (Hansen and Zilberstein 1998), expand through the state space using the supplied model of the MDP. In contrast stochastic, or Monte-Carlo, approaches sample trajectories from the model and use statistics gathered from those samples to choose a policy (Kocsis and Szepesvári 2006).

Cross-References

- ▶ Bayesian Network
- ▶ Curse of Dimensionality
- ▶ Markov Chain Monte Carlo
- ▶ Partially Observable Markov Decision Processes
- ▶ Reinforcement Learning
- ▶ Temporal Difference Learning

Recommended Reading

Albus JS (1981) Brains, behavior, and robotics. BYTE, Peterborough. ISBN:0070009759

Andre D, Friedman N, Parr R (1997) Generalized prioritized sweeping. In: Neural and information processing systems, Denver, pp 1001–1007

Andre D, Russell SJ (2002) State abstraction for programmable reinforcement learning agents. In: Proceedings of the eighteenth national conference on artificial intelligence (AAAI), Edmonton

Baird LC (1995) Residual algorithms: reinforcement learning with function approximation. In: Prieditis A, Russell S (eds) Machine learning: proceedings of the twelfth international conference (ICML95). Morgan Kaufmann, San Mateo, pp 30–37

Bellman RE (1957) Dynamic programming. Princeton University Press, Princeton

Bertsekas DP, Tsitsiklis J (1996) Neuro-dynamic programming. Athena Scientific, Belmont

Dietterich TG (2000) Hierarchical reinforcement learning with the MAXQ value function decomposition. J Artif Intell Res 13:227–303

Gordon GJ (1995) Stable function approximation in dynamic programming (Technical report CMU-CS-95-103). School of Computer Science, Carnegie Mellon University

Guestrin C et al (2003) Efficient solution algorithms for factored MDPs. J Artif Intell Res 19:399–468

Hansen EA, Zilberstein S (1998) Heuristic search in cyclic AND/OR graphs. In: Proceedings of the fifteenth national conference on artificial intelligence. http://rbr.cs.umass.edu/shlomo/papers/HZaaai98.html

Howard RA (1960) Dynamic programming and Markov processes. MIT Press, Cambridge

Kocsis L, Szepesvári C (2006) Bandit based Monte-Carlo planning. In: European conference on machine learning (ECML), Berlin. Lecture notes in computer science, vol 4212. Springer, pp 282–293

Moore AW, Atkeson CG (1993) Prioritized sweeping: reinforcement learning with less data and less real time. Mach Learn 13:103–130

Moore AW, Baird L, Pack Kaelbling L (1999) Multi-value-functions: efficient automatic action hierarchies for multiple goal MDPs. In: International joint conference on artificial intelligence (IJCAI99), Stockholm

Munos R, Moore AW (2001) Variable resolution discretization in optimal control. Mach Learn 1:1–31

Puterman ML (1994) Markov decision processes: discrete stochastic dynamic programming. Wiley series in probability and mathematical statistics. Applied probability and statistics section. Wiley, New York. ISBN:0-471-61977-9

St-Aubin R, Hoey J, Boutilier C (2000) APRICODD: approximate policy construction using decision diagrams. In: NIPS-2000, Denver

Sutton RS, Precup D, Singh S (1998) Intra-option learning about temporally abstract actions. In: Machine learning: proceedings of the fifteenth international conference (ICML98). Morgan Kaufmann, Madison, pp 556–564

Tsitsiklis JN, Van Roy B (1997) An analysis of temporal-difference learning with function approximation. IEEE Trans Autom Control 42(5):674–690

Markov Model

▶ Markov Process

Markov Net

▶ Markov Network

Markov Network

Synonyms

Markov net; Markov random field

Definition

A Markov network is a form of undirected ▶ graphical model for representing multivariate probability distributions.

Cross-References

▶ Graphical Models

Markov Process

Synonyms

Markov chain; Markov model

A stochastic process in which the conditional probability distribution of future states of the process, given the present state and all past states, depends only upon the present state. A process with this property may be called Markovian. The best known Markovian processes are *Markov chains*, also known as *Markov Models*, which are discrete-time series of states with transition probabilities. Markov chains are named after Andrey Markov (1865–1922), who introduced several significant new notions to the concept of stochastic processes. Brownian motion is another well-known phenomenon that, to close approximation, is a Markov process.

Recommended Reading

Meyn SP, Tweedie RL (1993) Markov chains and stochastic stability. Springer, London

Markov Random Field

▶ Markov Network

Markovian Decision Rule

Synonyms

Randomized decision rule

Definition

In a ▶ Markov decision process, a *decision rule*, d_t, determines what *action* to take, based on the history to date at a given *decision epoch* and for any possible *state*. It is *deterministic* if it selects a single member of $A(s)$ with probability 1 for each $s \in S$ and for a given h_t, and it is *randomized* if it selects a member of $A(s)$ at random with probability $q_{d_t(h_t)}(a)$. It is *Markovian* if it depends on h_t only through s_t. That is, $d_t(h_t) = d_t(s_t)$.

Maxent Models

▶ Maximum Entropy Models for Natural Language Processing

Maximally General Hypothesis

▶ Most General Hypothesis

Maximally Specific Hypothesis

▶ Most Specific Hypothesis

Maximum Entropy Models for Natural Language Processing

Adwait Ratnaparkhi
Yahoo!, Sunnyvale, CA, USA

Abstract

This chapter provides an overview of the maximum entropy framework and its application to a problem in natural language processing. The framework provides a way to combine many pieces of evidence from an annotated training set into a single probability model. The framework has been applied to many tasks in natural language processing, including part-of-speech tagging. This chapter covers the maximum entropy formulation, its relationship to maximum likelihood, a parameter estimation method, and the details of the part-of-speech tagging application.

Synonyms

Maxent models; Log-linear models; Statistical natural language processing

Definition

The term maximum entropy refers to an optimization framework in which the goal is to find the probability model that maximizes entropy over the set of models that are consistent with the observed evidence.

The information-theoretic notion of entropy is a way to quantify the uncertainty of a probability model; higher entropy corresponds to more uncertainty in the probability distribution. The rationale for choosing the maximum entropy model – from the set of models that meet the evidence – is that any other model assumes evidence that has not been observed (Jaynes 1957).

In most natural language processing problems, observed evidence takes the form of co-occurrence counts between some prediction of interest and some linguistic context of interest. These counts are derived from a large number of linguistically annotated examples, known as a corpus. For example, the frequency in a large corpus with which the word *that* co-occurs with the tag corresponding to determiner, or *DET*, is a piece of observed evidence. A probability model is consistent with the observed evidence if its calculated estimates of the co-occurrence counts agree with the observed counts in the corpus.

The goal of the maximum entropy framework is to find a model that is consistent with the co-occurrence counts, but is otherwise maximally uncertain. It provides a way to combine many pieces of evidence into a single probability model. An iterative parameter estimation procedure is usually necessary in order to find the maximum entropy probability model.

Motivation and Background

The early 1990s saw a resurgence in the use of statistical methods for natural language processing (Church and Mercer 1993). In particular, the IBM TJ Watson Research Center was a prominent advocate in this field for statistical methods such as the maximum entropy framework. Language modeling for speech recognition (Lau et al. 1993) and machine translation (Berger et al. 1996) were among the early applications of this framework.

Structure of Learning System

The goal of a typical natural language processing application is to automatically produce linguistically motivated categories or structures over freely occurring text. In statistically based approaches, it is convenient to produce the categories with a conditional probability model p such that $p(a|b)$ is the probability of seeing a prediction of interest a (e.g., a part-of-speech tag) given a linguistic context of interest b (e.g., a word).

The maximum entropy framework discussed here follows the machine learning approach to

NLP, which assumes the existence of a large corpus of linguistically annotated examples. This annotated corpus is used to create a training set, which in turn is used to estimate the probability model p.

Representing Evidence

Evidence for the maximum entropy model is derived from the training set. The training set is a list of (prediction, linguistic context) pairs that are generated from the annotated data. However, in practice, we do not record the entire linguistic context. Instead, linguistically motivated Boolean-valued questions reduce the entire linguistic context to a vector of question identifiers. Therefore, each training sample looks like:

Prediction	Question vector
a	$q_1 \dots q_n$

where a is the prediction and where $q_1 \dots q_n$ is a vector of questions that answered `true` for the linguistic context corresponding to this training sample. The questions must be designed by the experimenter in advance, and are specifically designed for the annotated data and the problem space.

In the framework discussed here, any piece of evidence is represented with a *feature*. A feature correlates a prediction a with an aspect of a linguistic context b, captured by some question:

$$f_j(a,b) = \begin{cases} 1 \text{ if } a = x \text{ and } q(b) = \text{true} \\ 0 \text{ otherwise} \end{cases}$$

Combining the Evidence

The maximum entropy framework provides a way to combine all the features into a probability model. In the conditional maximum entropy formulation (Berger et al. 1996), the desired model p^* is given by:

$$P = \{p | E_p f_j = E_{\tilde{p}} f_j, j = 1 \dots k\} \quad (1)$$

$$H(p) = -\sum_{a,b} \tilde{p}(b) p(a|b) \log p(a|b)$$

$$p^* = \text{argmax}_{p \in P} H(p)$$

where $H(p)$ is the conditional entropy of p, $\tilde{p}(b)$ is the observed probability of the linguistic context b in the training set, and P is the set of models that are consistent with the observed data. A model p is consistent if its own feature expectation $E_p f_j$ is equal to the observed feature expectation $E_{\tilde{p}} f_j$, for all $j = 1 \dots k$ features. $E_{\tilde{p}} f_j$ can be interpreted as the observed count of f_j in the training sample, normalized by the training sample size. Both are defined as follows:

$$E_p f_j = \sum_{a,b} \tilde{p}(b) p(a|b) f_j(a,b)$$

$$E_{\tilde{p}} f_j = \sum_{a,b} \tilde{p}(a,b) f_j(a,b)$$

According to the maximum entropy framework, the optimal model p^* is the most uncertain model among those that satisfy the feature constraints. It is possible to show that the form of the optimal model must be log-linear:

$$p^*(a|b) = \frac{1}{Z(b)} \prod_{j=1\dots k} \alpha_j^{f_j(a,b)} \quad (2)$$

$$Z(b) = \sum_{a'} \prod_{j=1\dots k} \alpha_j^{f_j(a',b)}$$

Here $Z(b)$ is a normalization factor, and $\alpha_j > 0$. Each model parameter α_j can be viewed as the "strength" of its corresponding feature f_j; the conditional probability is the normalized product of the feature weights of the active features.

Relationship to Maximum Likelihood

The maximum entropy framework described here has an alternate interpretation under the more commonly used technique of maximum likelihood estimation.

$$Q = \left\{ p | p(a|b) = \frac{1}{Z(b)} \prod_{j=1\dots k} \alpha_j^{f_j(a,b)} \right\}$$

$$L(p) = \sum_{a,b} \tilde{p}(a,b) \log p(a|b)$$

$$q^* = \underset{p \in Q}{\text{argmax}} L(p)$$

Here Q is the set of models of form (2), $\tilde{p}(a,b)$ is the observed probability of prediction a together with linguistic context b, $L(p)$ is the log-likelihood of the training set, and q^* is the maximum likelihood model. It can be shown that $p^* = q^*$; maximum likelihood estimation for models of the form (2) gives the same answer as maximum entropy estimation over the constraints on feature counts (1). The difference between approaches is that the maximum likelihood approach assumes the form of the model, whereas the maximum entropy approach assumes the constraints on feature expectations, and *derives* the model form.

Parameter Estimation

The Generalized Iterative Scaling (GIS) algorithm (Darroch and Ratcliff 1972) is the easiest way to estimate the parameters for this kind of model. The iterative updates are given below:

$$\alpha_j^{(0)} = 1$$

$$\alpha_j^{(n)} = \alpha_j^{(n-1)} \left[\frac{E_{\tilde{p}} f_j}{E_p f_j} \right]^{\frac{1}{C}}$$

GIS requires the use of a "correction" feature g and constant $C > 0$, which are defined so that $g(a,b) = C - \sum_{j=1\ldots k} f_j(a,b)$ for any (a,b) pair in the training set. Normally, the correction feature g must be trained in the model along with the k original features, although (Curran and Clark 2003) show that GIS converges even without the correction feature. The number of iterations needed to achieve convergence depends on certain aspects of the data, such as the training sample size and the feature set size, and is typically tuned for the problem at hand.

Other algorithms for parameter estimation include the Improved Iterative Scaling (Berger et al. 1996) algorithm and the Sequential Conditional GIS (Goodman 2002) algorithm. The list given here is not complete; many other numerical algorithms can be applied to maximum entropy parameter estimation, see Malouf (2002) for a comparison.

It is usually difficult to assess the reliability of features that occur infrequently in the training set, especially those that occur only once. When the parameters are trained from low frequency feature counts, maximum entropy models – as well as many other statistical learning techniques – have a tendency to "overfit" the training data. In this case, performance on training data appears very high, but performance on the intended test data usually suffers. *Smoothing* or *regularization* techniques are designed to alleviate this problem for statistical models; some smoothing techniques for maximum entropy models are reviewed in Chen and Rosenfeld (1999).

Applications

This framework has been used as a generic machine learning toolkit for many problems in natural language processing. Like other generic machine learning techniques, the core of the maximum entropy framework is invariant across different problem spaces. However, some information is specific to each problem space:

Predictions: The space of predictions for this model

Questions: The space of questions for this model

Feature Selection: Any possible (question, prediction) pair can be used as a feature. In complex models, only a small subset of all the possible features are used in a model. The feature selection strategy specifies how to choose the subset.

For a given application, it suffices to give the above three pieces of information to fully specify a maximum entropy probability model.

Part-of-Speech Tagging

Part-of-speech tagging is a well-known task in computational linguistics in which the goal is to disambiguate the part-of-speech of all the words in a given sentence. For example, it can be nontrivial for a computer to disambiguate the part-of-speech of the word *flies* in the following famous examples:

- Fruit *flies* like a banana.
- Time *flies* like an arrow.

The word *flies* behaves like a noun in the first case, and like a *verb* in the second case. In the machine learning approach to this problem, co-occurrence statistics of tags and words in the linguistic context are used to create a predictive model for part-of-speech tags.

The computational linguistics community has created annotated corpora to help build and test algorithms for tagging. One such corpus, known as the Penn treebank (Marcus et al. 1994), has been used extensively by machine learning and statistical NLP practitioners for problems like tagging. In this corpus, roughly 1 M words from the Wall St. Journal have manually been assigned part-of-speech tags. This corpus can be converted into a set of training samples, which in turn can be used to train a maximum entropy model.

Model Specification

For tagging, the goal is a maximum entropy model p that will produce a probability of seeing a tag at position i, given the linguistic context of the ith word, the surrounding words, and the previously predicted tags, written as $p(t_i|t_{i-1}\dots t_1, w_1\dots w_n)$. The intent is to use the model left-to-right, one word at a time. The maximum entropy model for tagging (Ratnaparkhi 1996) is specified as:

Predictions: The 45 part-of-speech tags of the Penn treebank

Questions: Listed below are the questions and question patterns. A question pattern has a placeholder variable (e.g., X, Y) that is instantiated by scanning the annotated corpus for examples in which the patterns match. Let i denote the position of the current word in the sentence, and let w_i and t_i denote the word and tag at position i, respectively.

- Does $w_i = X$?
- Does $w_{i-1} = X$?
- Does $w_{i-2} = X$?
- Does $w_{i+1} = X$?
- Does $w_{i+2} = X$?
- Does $t_{i-1} = X$?
- Does $t_{i-1}t_{i-2} = X, Y$?
- For word that occur less than 5 times in the training set:
  - Are the first K (for $K \leq 4$) characters $X_1 \dots X_K$?
  - Are the last K (for $K \leq 4$) characters $X_1 \dots X_K$?
  - Does the current word contain a number?
  - Does the current word contain a hyphen?
  - Does the current word contain an uppercase character?

Feature Selection: Any feature whose count in the training data is less than 10 is discarded.

While the features for each probability decision could in theory look at the entire linguistic context, they actually only look at a small window of words surrounding the current word, and a small window of tags to the left. Therefore each decision effectively makes the markov-like assumption given in Eq. (3).

$$p(t_i|t_{i-1}\dots t_1, w_1\dots w_n)$$

$$= p(t_i|t_{i-1}t_{i-2}w_{i-2}w_{i-1}w_iw_{i+1}w_{i+2}) \qquad (3)$$

$$= \frac{\prod_{j=1\dots k}\alpha_j^{f_j(t_i, t_{i-1}t_{i-2}w_{i-2}w_{i-1}w_iw_{i+1}w_{i+2})}}{Z(t_{i-1}t_{i-2}w_{i-2}w_{i-1}w_iw_{i+1}w_{i+2})} \qquad (4)$$

Equation (4) is the maximum entropy model for tagging. Each conditional probability of a prediction t_i given some context $t_{i-1}t_{i-2}w_{i-2}w_{i-1}w_i$ $w_{i+1}w_{i+2}$ is the product of the features that are active for that (prediction, context) pair.

Training Data

The training set is created by applying the questions to each word in the training set. For example, when scanning the word *flies* in the sentence "Time flies like an arrow" the training example would be:

Prediction	Question vector
$verb$	w_i = flies, w_{i-1} = Time, w_{i-2} = ∗bd∗, w_{i+1} = like, w_{i+2} = an, t_{i-1} = *noun*, $t_{i-1}t_{i-2}$ = *noun*, ∗bd∗

Here *bd* is a special symbol for boundary. The tags have been simplified for this example; the actual tags in the Penn treebank are more fine-grained than *noun* and *verb*.

Hundreds of thousands of training samples are used to create candidate features. Any possible (prediction, question) pair that occurs in training data is a candidate feature. The feature selection strategy is a way to eliminate unreliable or noisy features from the candidate set. For the part-of-speech model described here, a simple frequency threshold is used to implement feature selection.

Given a selected feature set, the GIS algorithm is then used to find the optimal value for the corresponding α_j parameters. For this application, roughly 50 iterations of GIS sufficed to achieve convergence.

Search for Best Sequence

The probability model described thus far will produce a distribution over tags, given a linguistic context including and surrounding the current word. In practice we need to tag entire sentences, which means that the model must produce a *sequence* of tags. Tagging is typically performed left-to-right, so that each decision has the left context of previously predicted tags. The probability of the best tag sequence for an n-word sentence is factored as:

$$\begin{aligned} &p(t_1 \dots t_n | w_1 \dots w_n) \\ &= \prod_{i=1\dots n} p(t_i | t_{i-1} \dots t_1, w_1 \dots w_n) \end{aligned}$$

The desired tag sequence is the one with the highest conditional sequence probability:

$$t_1^* \dots t_n^* = \operatorname*{argmax}_{t_1 \dots t_n} p(t_1 \dots t_n | w_1 \dots w_n)$$

A dynamic programming procedure known as the Viterbi algorithm is typically used to find the highest probability sequence.

Other NLP Applications

Other NLP applications have used maximum entropy models to predict a wide variety of linguistic structure. The statistical parser in Ratnaparkhi (1999) uses separate maximum entropy models for part-of-speech, chunk, and parse structure prediction. The system in Borthwick (1999) uses maximum entropy models for named entity detection, while the system in Ittycheriah et al. (2001) uses them as sub-components for both answer type prediction and named entity detection. Typically, such applications do not need to change the core framework, but instead need to modify the meaning of the predictions, questions, and feature selection to suit the intended task of the application.

Future Directions

Conditional random fields (Lafferty et al. 2001), or CRFs, are an alternative to maximum entropy models that address the *label bias* issue. Label bias affects sequence models that predict one element at a time, in which features at a given state (or word, in the case of POS tagging) compete with each other, but do not compete with features at any other state in the sequence. In contrast, a CRF model directly produces a probability distribution over the entire sequence, and therefore allows global competition of features across the entire sequence. The parameter estimation for CRFs is related to the Generalized Iterative Scaling algorithm used for maximum entropy models. See Sha and Pereira (2003) for a example of CRFs applied to noun phrase chunking.

Another recently published future direction is Collobert et al. (2011), which presents a multilayer neural network approach for several sequence labeling tasks, including POS tagging. This approach avoids task-specific feature engineering – like the questions in section "Model Specification" – and instead uses the neural network training algorithm to discover internal representations for the word and tag context. It also uses large amounts of unlabeled data to enhance the internal representations for words.

Recommended Reading

Berger AL, Della Pietra SA, Della Pietra VJ (1996) A maximum entropy approach to natural language processing. Comput Linguist 22(1):39–71

Borthwick A (1999) A maximum entropy approach to named entity recognition. PhD thesis, New York University

Chen S, Rosenfeld R (1999) A Gaussian prior for smoothing maximum entropy models. Technical report CMUCS-99-108, Carnegie Mellon University

Church KW, Mercer RL (1993) Introduction to the special issue on computational linguistics using large corpora. Comput Linguist 19(1):1–24

Collobert R, Weston J, Bottou L, Karlen M, Kavukcuoglu K, Kuksa P (2011) Natural language processing (almost) from scratch. J Mach Learn Res 12:2493–2537

Curran JR, Clark S (2003) Investigating GIS and smoothing for maximum entropy taggers. In: Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 1. Association for Computational Linguistics, pp 91–98

Darroch J, Ratcliff D (1972) Generalized iterative scaling for log-linear models. Ann Stat 43(5):1470–1480

Goodman J (2002) Sequential conditional generalized iterative scaling. In: Proceedings of the Association for Computational Linguistics

Ittycheriah A, Franz M, Zhu W, Ratnaparkhi A (2001) Question answering using maximum-entropy components. In: Procedings of NAACL

Jaynes ET (1957) Information theory and statistical mechanics. Phys Rev 106(4):620–630

Lafferty J, McCallum A, Pereira F (2001) Conditional random fields: probabilistic models for segmenting and labeling sequence data. In: Proceedings of the 18th international conference on machine learning. Morgan Kaufmann, San Francisco, pp 282–289

Lau R, Rosenfeld R, Roukos S (1993) Adaptive language modeling using the maximum entropy principle. In: Proceedings of the ARPA human language technology workshop. Morgan Kaufmann, San Francisco, pp 108–113

Malouf R (2002) A comparison of algorithms for maximum entropy parameter estimation. In: Sixth conference on natural language learning, pp 49–55

Marcus MP, Santorini B, Marcinkiewicz MA (1994) Building a large annotated corpus of English: the Penn Treebank. Comput Linguist 19(2): 313–330

Ratnaparkhi A (1996) A maximum entropy model for part-of-speech tagging. In: Brill E, Church K (eds) Proceedings of the conference on empirical methods in natural language processing. Association for Computational Linguistics, Somerset, pp 133–142

Ratnaparkhi A (1999) Learning to parse natural language with maximum entropy models. Mach Learn 34(1–3):151–175

Sha F, Pereira F (2003) Shallow parsing with conditional random fields. In: Proceedings of HLT-NAACL, pp 213–220

McDiarmid's Inequality

Synonyms

Bounded differences inequality

Definition

McDiarmid's inequality shows how the values of a bounded function of independent random variables concentrate about its mean. Specifically, suppose $f : \mathcal{X}^n \to R$ satisfies the bounded differences property. That is, for all $i = 1, \ldots, n$ there is a $c_i \geq 0$ such that for all $x_1, \ldots, x_n, x' \in \mathcal{X}$

$$\begin{aligned}&|f(x_1,\ldots,x_n)\\&\quad - f(x_1,\ldots,x_{i-1},x',x_{i+1},\ldots,x_n)| \leq c_i .\end{aligned}$$

If $\mathbf{X} = (X_1, \ldots, X_n) \in \mathcal{X}^n$ is a random variable drawn according to P^n and $\mu = E_{P^n}[f[\mathbf{X}]$ then for all $\epsilon > 0$

$$P^n(f(\mathbf{X}) - \mu \geq \epsilon) \leq \exp\left(\frac{2\epsilon^2}{\sum_{i=1}^n c_i^2}\right).$$

McDiarmid's is a generalization of Hoeffding's inequality, which can be obtained by assuming $\mathcal{X} = [a,b]$ and choosing $f(\mathbf{X}) = \sum_{i=1}^n X_i$. When applied to empirical risks this inequality forms the basis of many ▶ generalization bounds.

MCMC

▶ Markov Chain Monte Carlo

Mean Absolute Deviation

▶ Mean Absolute Error

Mean Absolute Error

Synonyms

Absolute error loss; Mean absolute deviation; Mean error

Definition

Mean Absolute Error is a ▶ model evaluation metric used with regression models. The mean absolute error of a model with respect to a ▶ test set is the mean of the absolute values of the individual prediction errors on over all ▶ instances in the ▶ test set. Each prediction error is the difference between the true value and the predicted value for the instance.

$$mae = \frac{\sum_{i=1}^{n} abs(y_i - \lambda(x_i))}{n}$$

where y_i is the true target value for test instance x_i, $\lambda(x_i)$ is the predicted target value for test instance x_i, and n is the number of test instances.

Cross-References

▶ Mean Squared Error

Mean Error

▶ Mean Absolute Error

Mean Shift

Xin Jin[1] and Jiawei Han[2]
[1]PayPal Inc., San Jose, CA, USA
[2]University of Illinois at Urbana-Champaign, Urbana, IL, USA

Abstract

Mean Shift is a clustering algorithm based on kernel density estimation. Various extensions have been proposed to improve speed and quality.

Synonyms

Density estimator

Definition

Mean shift (Comaniciu and Meer 2002) is a nonparametric algorithm for partitional clustering which does not require specifying the number of clusters and can form any shape of clusters. The mean shift procedure was originally proposed by Fukunaga and Hostetler (1975). Cheng (1995) adapted it for image analysis. Comaniciu, Meer, and Ramesh presented the mean shift approach to solve low-level vision problems: image segmentation (Comaniciu and Meer 2002), adaptive smoothing (Comaniciu and Meer 2002), and kernel-based object tracking (Comaniciu et al. 2003).

Given n data points $\mathbf{x}_i$, $i = 1, \ldots, n$ in the d-dimensional space R^d, the multivariate kernel density estimator obtained with kernel $K(\mathbf{x})$ and window radius h is given by

$$f(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} K(\frac{\mathbf{x} - \mathbf{x}_i}{h}) \qquad (1)$$

Given the gradient of the density estimator, the mean shift is defined as the difference between the weighted (using the kernel as weights) mean and x, the center of the kernel,

$$\mathbf{m}_h(\mathbf{x}) = \frac{\sum_{i=1}^{n} \mathbf{x}_i g(||\frac{\mathbf{x}-\mathbf{x}_i}{h}||^2)}{\sum_{i=1}^{n} g(||\frac{\mathbf{x}-\mathbf{x}_i}{h}||^2)} - \mathbf{x} \qquad (2)$$

The mean shift vector is proportional to the normalized density gradient estimate, and thus points to the direction of the maximum increase in the density. By successively computing the mean shift vector and translating the kernel (window) by the vector, the mean shift procedure can guarantee converging at a nearby point where the gradient of density function is zero.

Extensions

There are many extensions to the mean shift algorithm. Methods have been proposed to improve the performance of mean shift on speed (Paris and Durand 2007) and on accuracy by adaptive bandwidths (Georgescu et al. 2003) and asymmetric kernels (Yilmaz 2007).

The mean shift algorithm is designed for static distributions; a modified algorithm called Continuously Adaptive Mean Shift (CAMSHIFT) (Bradski 1998) can deal with dynamically changing distributions, for example, the color probability distributions derived from video frame sequences.

Mean shift has been extended for manifold clustering. Subbarao and Meer (2006) and Tuzel et al. (2005) proposed extensions to Grassmann manifolds and Lie groups for motion segmentation and multibody factorization. The medoid shift (Sheikh et al. 2007) algorithm avoids the definition of a stopping criteria and performs data clustering on both linear and curved spaces. The quick shift (Vedaldi and Soatto 2008) algorithm was proposed to eliminate the over-fragmentation problem of medoid shift. Cetingul and Vidal (2009) proposed intrinsic mean shift for clustering on Stiefel and Grassmann manifolds. The approach presents an alternative mean shift formulation which performs the iterative optimization on the manifold of interest and intrinsically locates the modes via consecutive evaluations of a mapping.

Softwares

The following softwares have implementations of the mean shift clustering algorithm:

- Scikit-Learn. An open-source machine learning software written in Python. http://scikit-learn.org
- OpenCV. Open Source Computer Vision Library. Written in C/C++. http://opencv.org
- Apache Mahout. Open-source machine learning software in Java for use in Hadoop, with support on mean shift before version 0.8. http://mahout.apache.org
- ImageJ. A Java-based library for image analysis and processing. It has an image filtering plug-in using the mean shift filter. http://rsbweb.nih.gov/ij/plugins/mean-shift.html

Recommended Reading

Bradski GR (1998) Computer vision face tracking for use in a perceptual user interface. Intel Technol J Q2(Q2):214–219

Cetingul HE, Vidal R (2009) Intrinsic mean shift for clustering on stiefel and grassmann manifolds. In: IEEE conference on computer vision and pattern recognition (CVPR 2009), Miami, pp 1896–1902

Cheng Y (1995) Mean shift, mode seeking, and clustering. IEEE Trans Pattern Anal Mach Intell 17(8):790–799

Comaniciu D, Meer P (2002) Mean shift: a robust approach toward feature space analysis. IEEE Trans Pattern Anal Mach Intell 24(5):603–619

Comaniciu D, Ramesh V, Meer P (2003) Kernel-based object tracking. IEEE Trans Pattern Anal Mach Intell 25(5):564–577

Fukunaga K, Hostetler L (1975) The estimation of the gradient of a density function, with applications in pattern recognition. IEEE Trans Inf Theory 21(1):32–40

Georgescu B, Shimshoni I, Meer P (2003) Mean shift based clustering in high dimensions: a texture classification example. In: Proceedings of ninth IEEE international conference on computer vision 2003, Nice, vol 1, pp 456–463

Paris S, Durand F (2007) A topological approach to hierarchical segmentation using mean shift. In: IEEE conference on computer vision and pattern recognition (CVPR 2007), Minneapolis, MN, pp 1–8

Sheikh YA, Khan EA, Kanade T (2007) Mode-seeking by medoidshifts. In: IEEE 11th international conference on computer vision (ICCV 2007), Rio de Janeiro, pp 1–8

Subbarao R, Meer P (2006) Nonlinear mean shift for clustering over analytic manifolds. In: IEEE computer society conference on computer vision and pattern recognition (CVPR 2006), vol 1, pp 1168–1175

Tuzel O, Subbarao R, Meer P (2005) Simultaneous multiple 3d motion estimation via mode finding on lie groups. In: Tenth IEEE international conference on computer vision (ICCV 2005), vol 1, pp 18–25

Vedaldi A, Soatto S (2008) Quick shift and kernel methods for mode seeking. In: Forsyth D, Torr P, Zisserman A (eds) Computer vision ECCV 2008. Lecture notes in computer science, vol 5305. Springer, Berlin/Heidelberg, pp 705–718

Yilmaz A (2007) Object tracking by asymmetric kernel mean shift with automatic scale and orientation selection. In: IEEE conference on computer vision and pattern recognition (CVPR 2007), Minneapolis, MN, pp 1–6

Mean Squared Error

Synonyms

Quadratic loss; Squared error loss

Definition

Mean Squared Error is a ▸ model evaluation metric often used with ▸ regression models. The mean squared error of a model with respect to a ▸ test set is the mean of the squared prediction errors over all ▸ instances in the ▸ test set. The prediction error is the difference between the true value and the predicted value for an instance.

$$mse = \frac{\sum_{i=1}^{n}(y_i - \lambda(x_i))^2}{n}$$

where y_i is the true target value for test instance x_i, $\lambda(x_i)$ is the predicted target value for test instance x_i, and n is the number of test instances.

Cross-References

▸ Mean Absolute Error

Measurement Scales

Ying Yang
Australian Taxation Office, Box Hill, VIC,
Australia

Definition

Turning to the authority of introductory statistical textbooks (Bluman 1992; Samuels and Witmer 1999), there are two parallel ways to classify data into different types. Data can be classified into either categorical or ▸ numeric. Data can also be classified into different levels of ▸ measurement scales.

There are two parallel ways to classify data into different types. Data can be classified into either categorical or numeric. Data can also be classified into different *levels of* measurement scales.

Categorical versus Numeric

Variables can be classified as either categorical or numeric. **Categorical variables**, also often referred to as **qualitative variables**, are variables that can be placed into distinct categories according to some characteristics. Categorical variables sometimes can be arrayed in a meaningful rank order. But no arithmetic operations can be applied to them. Examples of categorical variables are

- Gender of a fish: male and female
- Student evaluation: fail, pass, good, and excellent

Numeric variables, also often referred to as *quantitative variables*, are numerical in nature. They can be ranked in order. They can also have meaningful arithmetic operations. Numeric variables can be further classified into two groups, discrete or continuous.

A *discrete variable* assumes values that can be counted. The variable cannot assume all values on the number line within its value range. An example of a discrete variable is *the number of children in a family*.

A *continuous variable* can assume all values on the number line within the value range. The values are obtained by measuring. An example of a continuous variable is *Fahrenheit temperature*.

Levels of Measurement Scales

In addition to being classified as either categorical or numeric, variables can also be classified by how they are categorized, counted, or measured. This type of classification uses measurement scales, and four common types of scales are used: nominal, ordinal, interval, and ratio.

The *nominal* level of measurement scales classifies data into mutually exclusive (nonoverlapping), exhaustive categories in which no order or ranking can be imposed on the data. An example of a nominal variable is *gender of a fish*: male and female.

The *ordinal* level of measurement scales classifies data into categories that can be ranked. However, the differences between the ranks cannot be calculated by arithmetic. An example of an ordinal variable is *student evaluation*, with values fail, pass, good, and excellent. It is meaningful to say that the student evaluation of pass ranks is higher than that of fail. It is not meaningful in the same way to say that the gender female ranks higher than the gender male.

The **interval** level of measurement scales ranks the data, and the differences between units of measure can be calculated by arithmetic. However, *zero* in the interval level of measurement means neither "nil" nor "nothing" as *zero* in arithmetic means. An example of an interval variable is *Fahrenheit temperature*. It is meaningful to say that the temperature A is two points higher than the temperature B. It is not meaningful in the same way to say that the student evaluation of pass is two points higher than that of fail. Besides, 0°F does not mean the absence of heat.

The **ratio** level of measurement scales possesses all the characteristics of interval measurement, and there exists a *zero* that, the same as arithmetic *zero*, means "nil" or "nothing." In consequence, true ratios exist between different units of measure. An example of a ratio variable is *number of children in a family*. It is meaningful to say that the number of children in the family A is twice that of the family B. It is not meaningful in the same way to say that the Fahrenheit temperature A is twice that of B.

The nominal level is the lowest level of measurement scales. It is the least powerful in terms of including data information. The ordinal level is higher. The interval level is even higher. The ratio level is the highest level. Any data conversion from a higher level of measurement scales to a lower level of measurement scales, such as ▸ discretization, will lose information. Table 1 gives a summary of the characteristics of different levels of measurement scales.

Measurement Scales, Table 1 Characteristics of different levels of measurement scales

Level	Ranking?	Arithmetic operation?	Arithmetic zero?
Nominal	No	No	No
Ordinal	Yes	No	No
Interval	Yes	Yes	No
Ratio	Yes	Yes	Yes

Summary

In summary, the following taxonomy applies to variable types:

- Categorical (qualitative) variables:
  Nominal
  Ordinal
- Numeric (quantitative) variables:
  Interval, either discrete or continuous
  Ratio, either discrete or continuous

Recommended Reading

Bluman AG (1992) Elementary statistics: a step by step approach. Wm. C. Brown Publishers, Dubuque

Samuels ML, Witmer JA (1999) Statistics for the life sciences, 2nd edn. Prentice-Hall Publishers, Upper Saddle River

Medicine: Applications of Machine Learning

Katharina Morik
Technische Universität Dortmund, Dortmund, Germany

Motivation

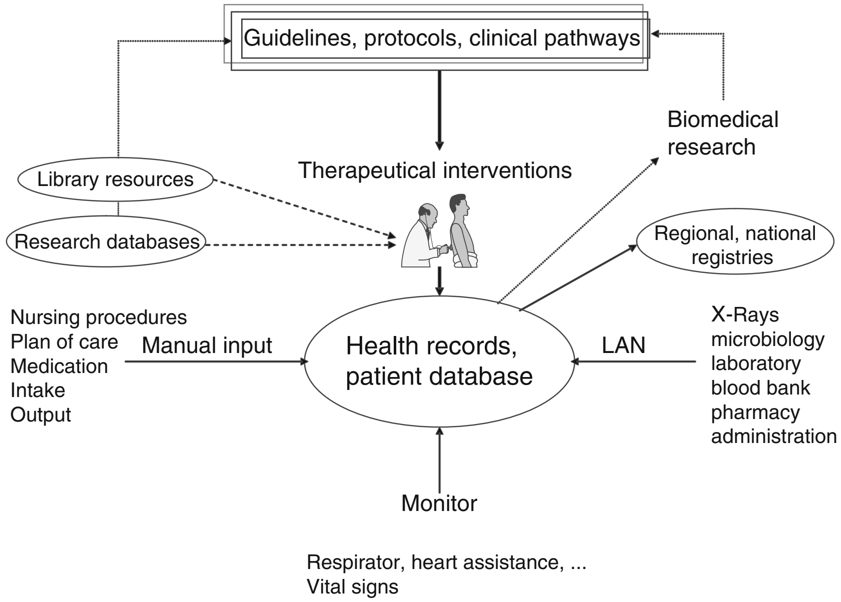
Health care has been an important issue in computer science since the 1960s. In addition to databases storing patient records, library resources (e.g., PubMed, a service of the U.S. National Library of Medicine that includes over 16 million citations from journals for biomedical articles back to the 1950s), administrative and financial systems, more sophisticated support of health care has been the aim of artificial intelligence (AI) from the very beginning on. Starting with expert systems which abstract laboratory findings and other vital parameters of a patient before they heuristically classify the patient into one of the modeled diagnoses (Shortliffe 1976), knowledge acquisition was discovered to be the bottleneck of systems for the automatic medical diagnosis. Machine learning came into play as a means of knowledge acquisition. Learning rules for (medical) expert systems focused on the heuristic classification step within expert systems. Given conveniently abstracted measurements of the patient's state, the classification was learned in terms of rules or ▶ decision trees. Since the early days, the use of machine learning for health care progressed in two ways:

- The abstraction of measurements of a patient's vital parameters is a learning task in its own right. Diverse kinds of data are to be handled: laboratory data, online measurements at the bedside, x-rays or other imaging data, genetic data,... Machine learning is confronted with a *diversity of representations* for the examples.
- Diagnosis is just one task in which physicians are to be supported. There are many more tasks which machine learning can ease. In intensive care, the addressee of the learning results can be a machine, e.g., the respirator. Financing health care and planning the medical resources (e.g., for a predicted epidemia) are yet another important issue. Machine learning is placed in a *diversity of medical tasks*.

The urgent need for sophisticated support of health care follows from reports which estimate up to 100,000 deaths in the USA each year due to medical error (Kohn et al. 2000).

Structure of the Problem

The overall picture of the medical procedures shows the kinds of data and how they are entered into the database of health records (a synonym is "patient database.") A monitoring system is given in intensive care units, which acquires ▶ time series from minute measurements. The observations at the bedside are entered manually into the system. The information from the hospital is entered via a local area network. The physician accesses information from libraries and research databases (dashed lines). Libraries, research databases, and biomedical research also influence the development of guidelines, protocols, and clinical pathways (dotted lines). Guidelines are rather abstract. Protocols of certain actions are integrated to become a clinical pathway which is a plan of both diagnostic and therapeutical actions for a typical patient with a specific diagnosis. The bold arrow shows the intended high-quality therapy. Guidelines and protocols promote evidence-based practices, reduce inter-clinician practice variations and support decision-making in patient care while constraining the costs of care. Computerized protocols can be generated based on guidelines. They have been proved useful in improving the quality and consistency of healthcare but the protocol development process is time-consuming (Ten Teije et al. 2006). This is where machine learning offers support. Usually, ontologies (e.g., in description logic) or other knowledge-based techniques (in medicine-specific formats like the Arden Syntax, GuideLine Interchange Format (GLIF), PROforma, Asbru, and EON) are used to support the development of protocols (de Clercq et al. 2004). By contrast, machine learning induces the current practices and their outcome from the health records (Smith et al. 2009). To reflect such use of Machine Learning, the bold arrows of the picture would need to be turned the other way around, protocols are learned from the data or evaluated based on the data. All (reversed) arrows mark possible applications of machine learning.



Diversity of Representations

The overall health record of a patient includes several types of data, not all of them are digital.

- Laboratory data consist of attributes almost always with numerical values, sometimes with discrete ordinal values, sometimes just binary values like “positive,” “negative.”
- Plain text states anamneses, diagnosis, and observations. From the text, key words can be transformed into attributes for machine learning.
- Online measurements at the bedside are time series. They are analyzed in order to find level changes or trends (Gather et al. 2006) and alarm the physician (Sieben and Gather 2007). In order to exploit the time series for further learning tasks, they often are abstracted (e.g., Bellazzi et al. 2002). Recently, online measurements from body sensors have raised attention in the context of monitoring patients at home (Amft and Tröster 2008).
- Sequences can also be considered time series, but the measurements are not equidistant and not restricted to numerical values. Examples are data gathered at doctors’ visits and long-term patient observations.
- X-rays or other imaging data (e.g., ultrasound imaging or more novel molecular imaging techniques like positron emission tomography, magnetic resonance imaging, or computer tomography) cannot be analyzed directly by machine learning algorithms. They require the extraction of features. It has been shown that the adequate extraction of features is more important than the selection of the best suited learning algorithm (Mavroforakis et al. 2006). The number of extracted features can become quite large. For instance, from 1,619 images of skin lesion, each 752 × 582 pixels, 107 features were extracted in order to detect melanoma using diverse learning algorithms (Dreiseitl et al. 2001). Hence, feature selection is also an important task in medical applications (Lucaces et al. 2009; Withayachumnankul et al. 2006). Often, different techniques are applied to gather data for the detection of the same disease. For instance, glaucoma detection uses standard automated perimetry or scanning laser or Heidelberg Retina Tomograph or stratus optical coherence tomography. It is not yet clear how important the choice among measurement types (devices) is with respect to feature extraction and machine learning.

- Tissue and blood: In vitro "data" also belong to health records. Immediately after biopsy or surgery, the tissue is transferred to the pathology department. After the pathologist has taken the sample needed for proper diagnosis, a representative tissue sample will be snap frozen and stored in liquid nitrogen or at −80 °C. Also blood cells are stored in a blood bank. From the specimen, the RNA is extracted and the so-called microarrays of gene expressions are developed and then scaled. The huge prognostic value of gene expression in patients with breast cancer has been shown by van't Veer et al. (2002). Genome research aims at revealing the impact of gene regulation and protein expression-regulation (taking into account the regulation of protein synthesis, protein ubiquitination, and post-translational modification) on, e.g., cancer diagnosis and response to therapies. Machine learning, particularly clustering, frequent itemset mining, and classification have been applied successfully (see learning from gene expression microarray data).

In addition to patient records, there are knowledge bases describing particular diseases or computerized protocols for particular therapies.

Medical Tasks

Diagnosis and Medication

Diagnosis is primarily a classification task. Given the description of the patient's state and a set of diseases, the learning algorithm outputs the classification into one of the classes. If physicians want to inspect the learned classifier, logic-based algorithms are preferred. Decision trees and the conceptual clustering algorithm AQ were used to diagnose breast cancer from nine abstracted descriptions like tumor size: 0–4, 5–9, ⋯ 50–54, 55–59 (Michalski et al. 1986; Cestnik et al. 1987).

▸ Bayesian methods were used to classify, e.g., diseases of the lymph node. Based on the examination of the extracted tissue, a pathologist enters the description. The Bayesian network (BN) outputs not only just one diagnosis, but the conditional probabilities for the diseases (Heckerman 1990). In particular, diagnosis for rather vague symptoms such as abdominal pain or lower back pain is well supported by BNs (McNaught et al. 2001). BNs are capable of incorporating given expert knowledge as priors. In order to combine textbook knowledge with empirical data, electronic literature was transformed into priors for BN structures. Then, from health records, the BN was learned as a model of ovarian tumors (Antal et al. 2004).

▸ Inductive logic programming (ILP) also allows to take into account background knowledge. This was used for an enhanced learning of medical diagnostic rules (Lavrac et al. 1993). The identification of glaucomatous eyes was effectively learned by ILP (Mizoguchi et al. 1997). One advantage of ILP is that the learned logic clauses can easily be integrated into a knowledge-based system and, hence, become operational for clinical practice.

Since some tests which deliver information about the patient's state can be costly – both, financially and in terms of a risk for the patient – ▸ cost-sensitive learning may be applied.

Since the error of classifying a patient as ill where he or she is not (false positives) is less harmful than classifying a patient as healthy where he or she is not (false negatives), the evaluation of the learning result most often is used in a biased way. The evaluation can be summarized in Table 1.

Precision is the proportion $\frac{A}{A+B}$, and *recall* is the proportion $\frac{A}{A+C}$. *Sensitivity* is synonymous to recall. In medical applications, sensitivity is balanced with respect to *specificity* being the proportion $\frac{B}{B+D}$ (synonym *false positives rate*). The analysis of the Receiver Operator Characteristic

Medicine: Applications of Machine Learning, Table 1
Evaluation measures

	True+	False−
Predicated+	A	B
Predicated−	C	D

(ROC) allows to evaluate learning according to sensitivity and specificity (see ▸ ROC analysis).

If not the understandability but only sensitivity and specificity are important, numerical learning algorithms are used to classify the patient's data. In particular, if the patient's state is described by numerical features, no discretization is necessary for numerical learners as is needed for the logic-based ones. Multilayer perceptrons (see ▸ Neural Networks), ▸ support vector machines (SVM), mixtures of Gaussians, and mixture of generalized Gaussian classifiers were trained on the numerical data of 189 normal eyes and 156 glaucomatous eyes (Goldbaum et al. 2002). The numerical description of the visual field is given by standard automated threshold perimetry. The medical standard procedure to interpret the visual field is to derive global indices. The authors compared performance of the classifiers with these global indices, using the area under the ROC curve. Two human experts were judged against the machine classifiers and the global indices by plotting their sensitivity–specificity pairs. The mixture of Gaussian had the greatest area under the ROC curve of the machine classifiers, and human experts were not better at classifying visual fields than the machine classifiers or the global indices.

Other approaches to glaucoma detection use different features describing the patient's state (Zangwill et al. 2004) or other numerical learners, e.g., ▸ logistic regression (Huang et al. 2006). For testing the learning from numerical attributes, the UCI Machine Learning Repository offers the arrhythmia database. The aim is to distinguish between the presence and absence of cardiac arrhythmia and to classify it in one of the 16 groups. About 279 attributes are given, 206 of them being numerical ones.

As has been shown in an application to intensive care, medication can be transformed into a set of classification tasks (Morik et al. 2000). Given measurements of eight vital signs, a decision is made for each of six drugs, whether to increase or to decrease it. This gives a set of classification tasks, which the ▸ SVM learned. Depending on the drug, the accuracy ranged from 81.3 % with 2.5 standard error to 86.9 % with 7 standard error. Additionally, on 41 cases, the SVM decision was compared with an expert's decisions when confronted with the same data. In 32 cases the expert chose the same direction of change as did the learned decision function. In 34 cases the learned decision was equal to the actual therapy. Another set of classification tasks were to decide every minute whether to increase, decrease, or leave the doses as it is. Again, each of these classifiers was learned by the SVM. From 1,319 examples decision functions were learned and tested on 473 examples. For training, an unbalanced cost function was used. The SVM cost factor for error was chosen according to $\frac{C_+}{C_-} = \frac{numberofnegativeexample}{numberofpositiveexample}$. The results again differed depending on the drug. For adrenaline, 79 % of the test cases were equally handled by the physician and the decision function. For adrenaline as well as for dobutamine, only in 1.5 % of the test cases the learned rule recommended the opposite direction of change. Again, a blind test with an expert showed that the learned recommendations' deviation from actual therapy was comparable to that of the human expert. Combining the two sets of classifications, for each minute and each patient, the support vector machine's decision function outputs a recommendation of treatment (Morik et al. 2000).

Prognosis and Quality of Care Assessment

Prognosis or outcome prediction is important for the evaluation of the quality of care provided. The standard statistical models use only a small set of covariates and a score variable, which indicates the severity of the illness. Machine learning may also rely on the aggregated score features, but is in addition capable of handling the features underlying the scores. Given health records of patients including the therapy, machine learning is to predict the outcome of care, e.g., classifies into mortal or surviving cases. The prediction of breast cancer survivability has been tested on a very large database comparing three learning methods (Delen et al. 2004). The results indicated that decision trees (here: C5) result in the best predictor with 93.6 % accuracy on the holdout sample (this prediction accuracy is better than any reported in the literature), artificial neural networks came out to be the second with 91.2 % accuracy, and the ▶ logistic regression models came out to be the worst of the three with 89.2 % accuracy.

Prediction of survival is a hard task for patients with serious stroke, because there is a long-term risk after the stay at the hospital. The scoring schemes (e.g., the Glasgow coma scale and the Ranking score) are not sufficient for predicting the outcome. In a data situation where 29 attributes (or features) were given for only 327 patient records, BNs were learned and compared with a handmade causal network. The results were encouraging – as soon as more electronic health records become available, the BNs will become closer to medical knowledge. Moreover, the discovery of relations on the basis of empirical data may enhance medical knowledge (Wu et al. 2001).

Carcinogenesis prediction was performed using ILP methods. As has become usual with cancer diagnosis and prognosis, there is a close link with microbiology (Srinivasan et al. 1994) (see Learning from gene expression microarray data).

Prognosis need not be restricted to mortality rates. In general, it is a means of quality assessment of clinical treatments. For instance, hemodialysis services have been assessed through temporal data mining by Bellazzi et al. (2002).

Finding subgroups of patients with devious reactions to a therapy might lead to a better understanding of a certain medical process (Atzmueller et al. 2005). While the before mentioned study aims at an enhanced expert – system interaction, a Dutch study aims at a more precise modeling of prognoses (Abu-Hanna and Lucas 2001). In an extensive study for eight different hospitals and 7,803 patients, two different models were combined: one for determining the subgroups and the other for building a model for each subgroup. For the prognoses of patients in an intensive care unit, subgroups have been detected using decision trees. The decision tree was trained to classify patients into the survival class and the mortality class on the basis of the nonaggregated features underlying the illness score. The leaves of the tree become subgroups. These are then used for training a logistic regression model of mortality based on the aggregated features.

Verification and Validation

Verification is the process of testing a model against a specification. In medicine, this often means to check clinical practice against expert protocols, or to check an actual diagnosis against one derived from textbook knowledge. Since many logic-based machine learning algorithms consist of a generalization and a specialization step, they can be used for verification. Generalization delivers rules from clinical data which can then be compared with given expert rules (protocols). Specialization is triggered by facts that contradict a learning hypothesis. Hence, using an expert rule as hypothesis, the learning algorithm counts the contradicting clinical cases and specializes the rule. For an early case study on verification and rule enhancement see, e.g., Morik et al. (1994). A more recent study compares a given clinical protocol for intensive care with actual therapies at another hospital (Scholz 2002). Decision trees and association rules have been learned in order to inspect and enhance the knowledge base of a web-based teledermatology system (Ou et al. 2007). While verification means to build the system right, validation means to build the right system. The borderline between verification and validation is fuzzy. On the one hand, medical practice is investigated with respect to the guidelines (verification), on the other hand, the guidelines are enhanced on the basis of medical practice (validation).

Moreover, learned models can be verified with respect to expert knowledge and validated with respect to clinical practice. A study on the hemodynamic monitoring of the critically ill integrated machine learning into a knowledge-based approach to evidence-based medicine. A knowledge base on drug effects was verified using patient records. Only 18 % of the observations showed vital signs of patients in the opposite direction than predicted by the knowledge base. Then, the knowledge base was used to validate therapeutical interventions proposed by a learned model. Accuracy measures of a model only reflect how well the learning result fits actual behavior of the physician and not how well it fits the "gold standard." Hence, a proposed intervention should be validated with respect to its effects on the patient. If the known effects push vital signs in the direction of the desired value range, the recommendation is considered sound, otherwise it is rejected. Using past data, the learned model was found to recommend an intervention with the desired effects in 81 % of the cases (Morik et al. 2002).

Intelligent Search in Medical Literature

Intelligent search in the overwhelming number of research publications supplies the information when it is needed. ILP has been successfully put to use for finding relevant medical documents (Dimec et al. 1999). Also the intelligent search in clinical free-text guidelines is an issue (Moskovitch et al. 2006). The techniques for text categorization can be applied to medical texts in the usual way. If the search engine not only labels the overall document but, in addition, phrases within it, the search could become more focused and also deliver paragraphs instead of complete texts. The biomedical challenge for *named entity recognition* requires the automatic extraction and classification of words referring to DNA, RNA, proteins, cell types, and cell lines from texts (Kim et al. 2004). Even more difficult is the discovery of medical knowledge from texts (Sanchez and Moreno 2005).

Epidemiology and Outbreak Detection

Understanding the transmission of infectious diseases and forecasting epidemics is an important task, since infections are distributed globally. Statistical approaches to spatio-temporal analysis of scan data are regularly used. There, a grid partitions the map into regions where occurrences of the disease are shown as points. "Hot spot" partitions are those of high density. By contrast, clustering detects hot spot regions depending on the data, hence, the shape of regions is flexible. Taking into account the a priori density of the population, a risk-adjusted nearest neighbor hierarchical clustering discovers "hot spot" regions. Also a risk-adjusted support vector machine with Gaussian kernel has successfully been applied to the problem of detecting regions with infectious disease outbreak. The discovery of hot spot regions can be exploited for predicting virus activity, if an indicator is known which can easily be observed. For instance, dead crows indicate activity of the West Nile virus. An overview of infectious disease informatics is given by Zeng et al. (2005).

Machine learning can also contribute to the understanding of the transmission of infectious diseases. A case study on tuberculosis epidemiology uses BNs to identify the distribution of tuberculosis patient attributes. The learning results captured the known statistical relationships. A relational model learned from the database directly using structured statistical models revealed several novel associations (Getoor et al. 2004).

Cross-References

- ▶ Class Imbalance Problem
- ▶ Classification
- ▶ Classifier Systems
- ▶ Cost-Sensitive Learning
- ▶ Decision Tree
- ▶ Feature Selection
- ▶ Inductive Logic Programming
- ▶ ROC Analysis
- ▶ Support Vector Machines
- ▶ Time Series

Recommended Reading

Abu-Hanna A, Lucas PJF (2001) Prognostic models in medicine: AI and statistical approaches [Editorial]. Methods Inf Med 40(1):1–5

Amft O, Tröster G (2008) Recognition of dietary events using on-body sensors. Artif Intell Med 42(2):121–136

Antal P, Fannes G, Timmerman D, Moreau Y, De Moor B (2004) Using literature and data to learn BNs as clinical models of ovarian tumors. Artif Intell Med 30(3): 257–281

Atzmueller M, Baumeister J, Hensing A, Richter E-J, Puppe F (2005) Subgroup mining for interactive knowledge refinement. In: Artificial intelligence in medicine (AIME). Springer, Berlin/Heidelberg, pp 453–462

Bellazzi R, Larizza C, Magni P, Bellazi R (2002) Quality assessment of dialysis services through intelligent data analysis and temporal data mining. In: Workshop at the 15th European conference on AI about intelligent data analysis in medicine and pharmacology, Lyon, pp 3–9

Cestnik B, Kononenko I, Bratko I (1987) ASSISTANT 86: a knowledge-elicitation tool for sophisticated users. In: Bratko I, Lavrac N (eds) Progress in machine learning. Sigma Press, Wilmslow, pp 31–45

de Clercq PA, Blomb JA, Korstenb HH, Hasman A (2004) Approaches for creating computer-interpretable guidelines that facilitate decision support. Artif Intell Med 31(1):1–27

Delen D, Walker G, Kadam A (2004) Predicting breast cancer survivability: a comparison of three data mining methods. Artif Intell Med 34(2):113–127

Dimec B, Dzeroski S, Todorovski L, Hristovski D (1999) WWW search engine for Slovenian and English medical documents. In: Proceedings of the 15th international congress for medical informatics. IOS Press, Amsterdam, pp 547–552

Dreiseitl S, Ohn-Machado L, Kittler H, Vinterbo S, Billhardt H, Binder M (2001) A comparison of machine learning methods for the diagnosis of pigmented skin lesions. J Biomed Inform 34:28–36

Gather U, Schettlinger K, Fried R (2006) Online signal extraction by robust linear regression. Comput Stat 21(1):33–51

Getoor L, Rhee JT, Koller D, Small P (2004) Understanding tuberculosis epidemiology using structured statistical models. Artif Intell Med 30(3):233–256

Goldbaum MH, Sample PA, Chan K, Williams J, Lee T-W, Blumenthal E et al (2002) Comparing machine learning classifiers for diagnosing glaucoma from standard automated perimetry. Investig Ophthalmol Vis Sci 43:162–169

Heckerman D (1990) Probabilistic similarity networks. Technical report STAN-CS-1316, Department of Computer Science and Medicine at Stanford

Huang ML, Chen HY, Hung PT (2006) Analysis of glaucoma diagnosis with automated classifiers using stratus optical coherence tomography. Opt Quantum Electron 37:1239–1249

Kim JD, Ohta T, Tsuruoka Y, Tateisi Y, Collier N (2004) Introduction to the bio-entity recognition task at JNLPBA. In: Collier N, Ruch P, Nazarenko A (eds) Proceedings of the international joint workshop on natural language processing in biomedicine and its applications. ACL, Morristown, pp 70–76

Kohn LT, Corrigan JM, Donaldson M (eds) (2000) To err is human – building a safer health system. National Academic Press, Washington, DC

Lavrac N, Dzeroski S, Prinat V, Krizman V (1993) The utility of background knowledge in learning medical diagnostic rules. Appl Artif Intell 7:273–293

Lucaces O, Taboada F, Albaiceta G, Domingues LA, Enriques P, Bahamonde A (2009) Predicting the probability of survival in intensive care unit patients from a small number of variables and training examples. Artif Intell Med 45(1):63–76

Mavroforakis M, Georgiou H, Dimitropoulos N, Cavouras D, Theodoridis S (2006) Mammographic masses characterization based on localized texture and dataset fractal analysis using linear, neural and support vector machine classifiers. Artif Intell Med 37(2):145–162

McNaught K, Clifford S, Vaughn M, Foggs A, Foy M (2001) A Bayesian belief network for lower back pain diagnosis. In: Lucas P, van der Gaag LC, Abu-Hanna A (eds) Bayesian models in medicine – Workshop at AIME, Caseais

Michalski R, Mozetic I, Hong J, Lavrac N (1986) The multi-purpose incremental learning system AQ15 and its testing application on three medical domains. In: Proceedings of the 5th national conference on artificial intelligence. Morgan Kaufmann, San Mateo, pp 1041–1045

Mizoguchi F, Ohwada H, Daidoji M, Shirato S (1997) Using inductive logic programming to learn classification rules that identify glaucomatous eyes. In: Lavraè N, Keravnou E, Zupan B (eds) Intelligent data analysis in medicine and pharmacology. Kluwer, Norwell, pp 227–242

Morik K, Imhoff M, Brockhausen P, Joachims T, Gather U (2000) Knowledge discovery and knowledge validation in intensive care. Artif Intell Med 19(3):225–249

Morik K, Joachims T, Imhoff M, Brockhausen P, Rüping S (2002) Integrating kernel methods into a knowledge-based approach to evidence-based medicine. In: Schmitt M, Teodorescu HN, Jain A, Jain A, Jain S, Jain LC (eds) Computational intelligence processing in medical diagnosis. Studies in fuzziness and soft computing, vol 96. Physica-Verlag, New York, pp 71–99

Morik K, Potamias G, Moustakis VS, Charissis G (1994) Knowledgeable learning using MOBAL: a medical case study. Appl Artif Intell 8(4):579–592

Moskovitch R, Cohen-Kashia S, Drora U, Levya I, Maimona A, Shahar Y (2006) Multiple hierarchical classification of free-text clinical guidelines. Artif Intell Med 37(3):177–190

Ou M, West G, Lazarescu M, Clay C (2007) Dynamic knowledge validation and verification for CBR teledermatology system. Artif Intell Med 39(1):79–96

Sanchez D, Moreno A (2005) Web mining techniques for automatic discovery of medical knowledge. In: Proceedings of the 10th conference on artificial intelligence in medicine, Aberdeen

Scholz M (2002) Using real world data for modeling a protocol for ICU monitoring. In: Lucas P, Asker L, Miksch S (eds) Working notes of the IDAMAP 2002 workshop, Lyon, pp 85–90

Shipp MA, Ross KN, Tamayo P, Weng AP, Kutok JL, Aguiar RC et al (2002) Diffuse large B-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning. Nat Med 8(1):68–74

Shortliffe EH (1976) Computer based medical consultations: MYCIN. Elsevier, New York/Amsterdam

Sieben W, Gather U (2007) Classifying alarms in intensive care–analogy to hypothesis testing. In: 11th conference on artificial intelligence in medicine (AIME). Springer, Berlin, pp 130–138

Smith WP, Doctor J, Meyer J, Kalet IJ, Philips MH (2009) A decision aid for intensity-modulated radiation-therapy plan selection in prostate cancer based on a prognostic Bayesian network and a Markov model. Artif Intell Med 46(2):119–130

Srinivasan A, Muggleton SH, King RD, Sternberg MJE (1994) Carcinogenesis prediction using inductive logic programming. In: Zupan B, Keravnou E, Lavrac N (eds) Intelligent data analysis in medicine and pharmacology. Kluwer, Norwell, pp 243–260

Ten Teije A, Lucas P, Miksch S (eds) (2006) Workshop on AI techniques in healthcare: evidence-based guidelines and protocols, held in conjunction with ECAI-2006, Riva del Garda

van't Veer LJ, Dai HY, van de Vijver MJ, He YDD, Hart AA, Mao M et al (2002) Gene expression profiling predicts clinical outcome of breast cancer. Nature 415:530–536

Withayachumnankul W, Ferguson B, Rainsford T, Findlay D, Mickan SP, Abbott D (2006) T-ray relevant frequencies for osteosarcoma classification. In: Abbott D, Kivshar YS, Rubinstein-Dunlop HH, Fan S-H (eds) Proceedings of SPIE, Brisbane

Wu X, Lucas P, Kerr S, Dijkhuisen R (2001) Learning Bayesian-network topologies in realistic medical domains. In: Intelligent data analysis in medicine and pharmacology. Medical Data Analysis. Springer, Berlin/Heidelberg, pp 302–307

Zangwill LM, Chan K, Bowd C, Hao J, Lee TW, Weinreb RN et al (2004) Heidelberg retina tomograph measurements of the optic disc and parapillary retina for detecting glaucoma analyzed by machine learning classifiers. Investig Ophthalmol Vis Sci 45(9):3144–3151

Zeng D, Chen H, Lynch C, Eidson M, Gotham I (2005) Infectious disease informatics and outbreak detection. In: Chen H, Fuller S, Friedman C, Hersh W (eds) Medical informatics: knowledge management and data mining in biomedicine. Springer, New York, pp 359–395

Memory-Based

▶ Instance-Based Learning

Memory-Based Learning

▶ Case-Based Reasoning

Merge-Purge

▶ Entity Resolution

▶ Record Linkage

Message

In ▶ Minimum Message Length inference, a binary sequence conveying information is called a message.

Meta-combiner

A *meta-combiner* is a form of ▶ ensemble learning technique used with ▶ missing attribute values. Its common topology involves base learners and classifiers at the first level, and meta-learner and meta-classifier at the second level. The meta-classifier combines the decisions of all the base classifiers.

Metaheuristic

Marco Dorigo[1], Mauro Birattari[1], and
Thomas Stützle[2]
[1]Université Libre de Bruxelles, Brussels,
Belgium
[2]Université libre de Bruxelles (ULB), Brussels,
Belgium

A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework that can be applied to different optimization problems with relatively few modifications. Examples of metaheuristics include simulated annealing, tabu search, iterated local search, evolutionary algorithms, and ant colony optimization.

Metalearning

Pavel Brazdil[2], Ricardo Vilalta[3], Christophe Giraud-Carrier[4], and Carlos Soares[1,2]
[1]LIAAD-INESC Porto L.A./Faculdade de Economia, University of Porto, Porto, Portugal
[2]LIAAD-INESC Tec/Faculdade de Economia, University of Porto, Porto, Portugal
[3]Department of Computer Science, University of Houston, Houston, TX, USA
[4]Department of Computer Science, Brigham Young University, Provo, UT, USA

Abstract

In the area machine learning / data mining many diverse algorithms are available nowadays and hence the selection of the most suitable algorithm may be a challenge. Tbhis is aggravated by the fact that many algorithms require that certain parameters be set. If a wrong algorithm and/or parameter configuration is selected, substandard results may be obtained. The topic of metalearning aims to facilitate this task. Metalearning typically proceeds in two phases. First, a given set of algorithms A (e.g. classification algorithms) and datasets D is identified and different pairs <ai,dj> from these two sets are chosen for testing. The dataset di is described by certain meta-features which together with the performance result of algorithm ai constitute a part of the metadata. In the second phase the metadata is used to construct a model, usually again with recourse to machine learning methods. The model represents a generalization of various base-level experiments. The model can then be applied to the new dataset to recommend the most suitable algorithm or a ranking ordered by relative performance. This article provides more details about this area. Besides, it discusses also how the method can be combined with hyperparameter optimization and extended to sequences of operations (workflows).

Synonyms

Adaptive learning; Dynamic selection of bias; Hyperparameter optimization; Learning to learn; Selection of algorithms, Ranking learning methods; Self-adaptive systems

Definition

Metalearning allows machine learning systems to benefit from their repetitive application. If a learning system fails to perform efficiently, one would expect the learning mechanism itself to adapt in case the same task is presented again. Metalearning differs from base learning in the scope of the level of adaptation; whereas learning at the base level is focused on accumulating experience on a specific task (e.g., credit rating, medical diagnosis, mine-rock discrimination, fraud detection, etc.), learning at the meta-level is concerned with accumulating experience on the performance of multiple applications of a learning system.

Briefly stated, the field of metalearning exploits the relation between tasks or domains and learning algorithms. Rather than starting afresh on each new task, metalearning facilitates evaluation and comparison of learning algorithms on many different previous tasks, establishes benefits and disadvantages, and then recommends the learning algorithm, or combination of algorithms, that maximizes some utility function on the new task. This problem can be seen as an instance of the algorithm selection task (Rice 1976).

The utility or usefulness of a given learning algorithm is often determined through a mapping between a characterization of the task and the algorithm's estimated performance (Brazdil and Henery 1994). In general, metalearning can recommend more than one algorithm. Typically, the number of recommended algorithms is significantly smaller than the number of all possible (available) algorithms (Brazdil et al. 2009).

Motivation and Background

The application of machine learning systems to classification and regression tasks has become a standard, not only in research but also in commerce and industry (e.g., finance, medicine, and engineering). However, most successful applications are custom designed, the result of skillful use of human expertise. This is due, in part, to the large, ever-increasing number of available machine learning systems, their relative complexity, and the lack of systematic methods for discriminating among them. The problem is further compounded by the fact that, in Knowledge Discovery from Databases, each operational phase (e.g., pre-processing, model generation) may involve a choice among various possible alternatives (e.g., progressive vs. random sampling, neural network vs. decision tree learning), as observed by Bernstein et al. (2005).

Current data mining systems are only as powerful as their users. These tools provide multiple algorithms within a single system, but the selection and combination of these algorithms must be performed before the system is invoked, generally by an expert user. For some researchers, the choice of learning and data transformation algorithms should be fully automated if machine learning systems are to be of any use to nonspecialists. Others claim that full automation of the data mining process is not within the reach of current technology. An intermediate solution is the design of assistant systems aimed at helping to select the right learning algorithm(s). Whatever the proposed solution, there seems to be an implicit agreement that metaknowledge should be integrated seamlessly into the data mining system. Metalearning focuses on the design and application of learning algorithms to acquire and use metaknowledge to assist machine learning users with the process of model selection. A general framework for this purpose, together with a survey of approaches, is in Smith-Miles (2008).

Metalearning is often seen as a way of redefining the space of inductive hypotheses searched by the learning algorithm(s). This issue is related to the idea of ▸ search bias, that is, search factors that affect the definition or selection of inductive hypotheses (Mitchell 1997). In this sense, metalearning studies how to choose the right bias dynamically and thus differs from base-level learning, where the bias is fixed or user parameterized. Metalearning can also be viewed as an important feature of self-adaptive systems, that is, learning systems that increase in efficiency through experience (Vilalta and Drissi 2002).

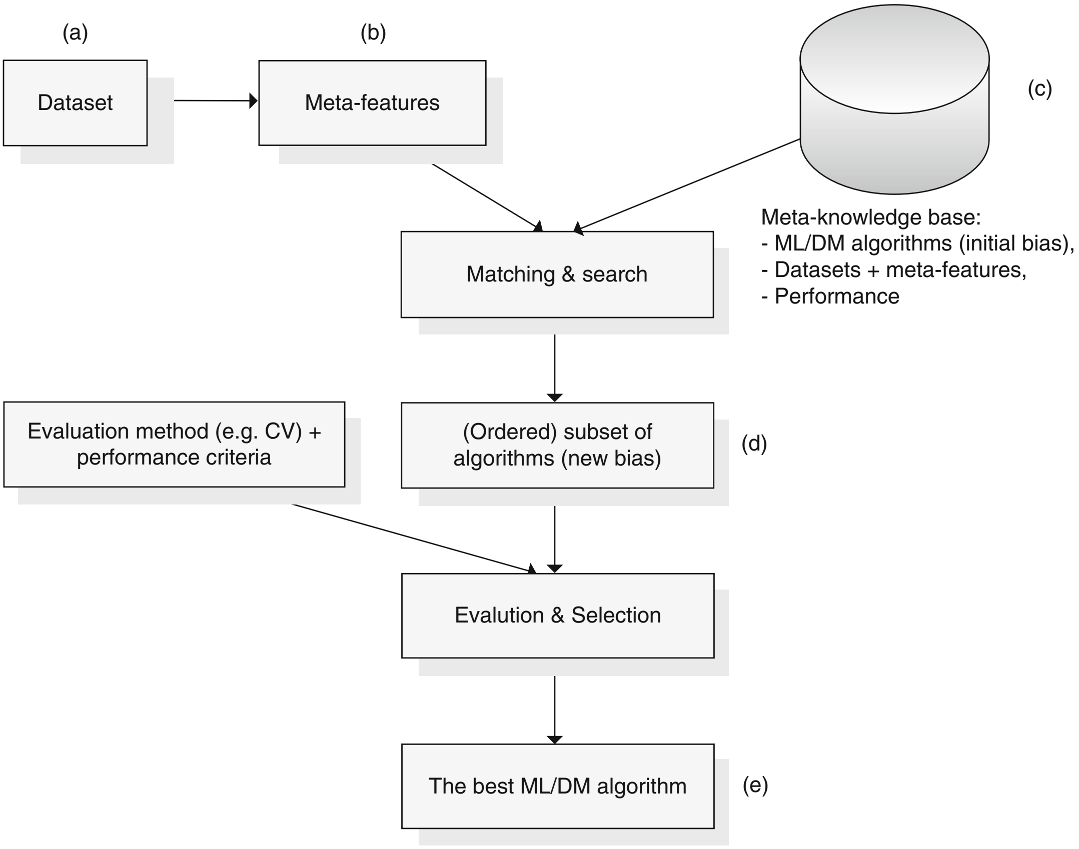
Structure of the Metalearning System

A metalearning system is essentially composed of two parts. One part is concerned with the acquisition of metaknowledge from machine learning systems. The other part is concerned with the application of metaknowledge to new problems with the objective of identifying an optimal learning algorithm or technique. The latter part – application of metaknowledge – can be used to help select or adapt suitable machine learning algorithms. So, for instance, if we are dealing with a ▸ classification task, metaknowledge can be used to select a suitable ▸ classifier for the new problem. Once this has been done, one can train the classifier and apply it to some unclassified sample for the purpose of class prediction.

In the following sections, we begin by describing scenarios corresponding to the case when metaknowledge has already been acquired. We then provide an explanation of how this knowledge is acquired.

Employing Metaknowledge to Select Machine Learning Algorithms

The aim of this section is to show that metaknowledge can be useful in many different settings. We will start by considering the problem of selecting suitable machine learning algorithms from a given set. The problem can be seen as a search problem. The search space includes the individual machine learning algorithms, and the aim is to identify the best algorithm. This process can be divided into two separate phases. In the first phase, the aim is to identify a suitable subset of machine learning algorithms based on an


(a)
Dataset
(b)
Meta-features
(c)
Meta-knowledge base:
- ML/DM algorithms (initial bias),
- Datasets + meta-features,
- Performance
Matching & search
Evaluation method (e.g. CV) + performance criteria
(Ordered) subset of algorithms (new bias)
(d)
Evalution & Selection
The best ML/DM algorithm
(e)

Metalearning, Fig. 1 Selection of machine learning algorithms: determining the reduced space and selecting the best alternative

input dataset (Fig. 1a–1b). The selection method used in this process can exploit metaknowledge (Fig. 1c). This is in general advantageous, as it often leads to better choices. In some work the result of this phase is represented in the form of a ranked subset of machine learning algorithms (Fig. 1d). The subset of algorithms represents the reduced bias space. The ranking (i.e., ordering of different algorithms) represents the procedural search bias.

The second phase is used to search through the reduced space. Each option is evaluated using a given performance criterion (e.g., accuracy). Typically, cross-validation is used to identify the best learning algorithm (Fig. 1e). We note that metaknowledge does not completely eliminate the need for the search process but rather provides a more effective search. The search effectiveness depends on the quality of metaknowledge.

Input to and Output from the Metalearning System

A metalearning approach to solving the algorithm selection problem relies on dataset characteristics or meta-features that provide some information to differentiate performance among a given set of learning algorithms. These include various types of measures, or meta-features, discussed in detail below.

Much previous work in dataset characterization has concentrated on extracting statistical and information-theoretic parameters estimated from the training set. Measures include the number of classes, the number of features, the ratio of examples to features, the degree of correlation between features and target concept, the average class entropy, etc. (Engels and Theusinger 1998). The disadvantage of this approach is that there is a limit to how much information these meta-features can capture, given that all these measures are uni- or bilateral measures only (i.e., they capture relationships between two attributes only or one attribute and the class).

Another approach is based on what are called *landmarkers*; these are simple and fast learners (Pfahringer et al. 2000). The accuracy of these simplified algorithms is used to characterize a dataset and to identify areas where each type of learner can be regarded as an expert. An interesting variation on the theme of landmarking uses information obtained on simplified versions of the data (e.g., samples). Accuracy results on these samples serve to characterize individual datasets and are referred to as *subsampling landmarks*.

In principle, any machine learning algorithm can be used at the meta-level. However, one important aspect of the metalearning task is the scarcity of training data. As a result, many researchers in the past have used *lazy learning* methods, such as k-NN, since these delay the generalization of metadata to the application phase (Nakhaeizadeh and Schnabl 1997). However, other types of models, such as neural networks, ranking trees, and bagging ensembles, have been proposed and proved rather successful (Sun and Pfahringer 2012, 2013).

There are several possible outputs or types of model a metalearning system can produce. Some focus on selecting the best algorithm in the set of available base learners; some attempt to predict the actual performance of individual algorithms; yet others assess the relative performance of different pairs of algorithms; finally, some systems produce a complete ranking of the base learners that can then be followed by minimal testing to identify the truly best algorithm for the user's dataset. One significant advantage of ranking methods is that they offer a next best alternative if the first algorithm seems to be suboptimal. As the set of base learners may contain variants of the same algorithms, and it would be wasteful to test them all before moving on to other types of algorithms, a recent approach known as active testing has been proposed, which seeks to identify the most promising algorithm that has a chance of surpassing the best algorithm identified so far (Leite et al. 2012).

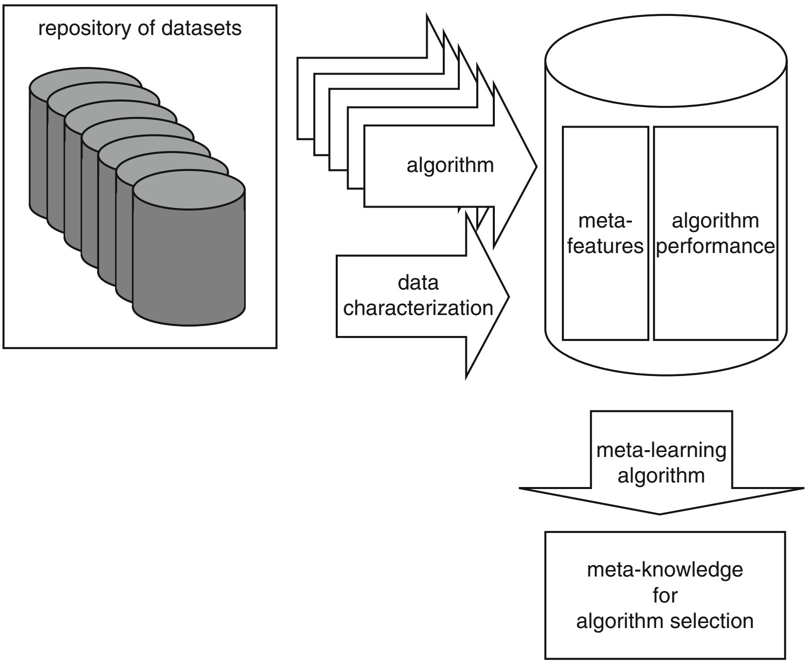
Acquisition of Metaknowledge

There are two natural ways in which metaknowledge can be acquired. One possibility is to rely on expert knowledge. Another possibility is to use an automatic procedure. We explore both alternatives briefly below.

One way of representing metaknowledge is in the form of rules that match domain (dataset) characteristics with machine learning algorithms. Such rules can be handcrafted, taking into account theoretical results, human expertise, and empirical evidence. For example, in decision tree learning, a heuristic rule can be used to switch from univariate tests to linear tests if there is a need to construct non-orthogonal partitions over the input space. This method has serious disadvantages however. First, the resulting rule set is likely to be incomplete. Second, timely and accurate maintenance of the rule set as new machine learning algorithms become available is problematic. As a result, most research has focused on automatic methods.

One other way of acquiring metaknowledge relies on automatic experimentation. For this we need a pool of problems (datasets) and a set of machine learning algorithms that we wish to consider. Then we need to define the experimental method that determines which alternatives we should experiment with and in which order (see Fig. 2 for details).

Suppose we have a dataset (characterized using certain meta-features), in combination with certain machine learning algorithms. The combination is assessed using an evaluation method (e.g., cross-validation) to produce performance results. The results, together with the characterization, represent a piece of metadata that is stored in the metaknowledge base. The process is then repeated for other combinations of datasets and algorithms.



Metalearning, Fig. 2 Acquisition of metadata for the metaknowledge base

Algorithm Selection and Hyperparameter Optimization

While this entry describes metalearning in the context of selecting algorithms for machine learning, there are a number of other areas, such as regression, time series forecasting, and optimization (Smith-Miles 2008), where algorithm selection is important and could benefit from a similar approach.

Similarly, there has been recent interest in the optimization community in the problem of hyperparameter optimization, wherein one seeks a set of hyperparameters for a learning algorithm, usually with the goal of obtaining good generalization and consequently low loss (Xu et al. 2008). Hyperparameter optimization is clearly relevant to algorithm selection, since most learning algorithms have parameters that can be adjusted and whose values may affect the performance of the learner. Historically, metalearning has largely ignored parameter selection, and hyperparameter optimization has largely ignored metalearning. Recent efforts in bringing the two fields together hold promise.

Applying Metalearning to Workflow Design for KDD

Much of the work in metalearning has focused on classification algorithm selection and thus addressed only a small fraction of the overall data mining process. In practice, users must not only select a classification learner but must often also consider various data pre-processing steps and other aspects of the process to build what are actually sequences of operations to apply to their data, also known as workflows. Several advances have been made in recent years in this area (Hilario et al. 2011; Kietz et al. 2012). Usually, it is possible to distinguish two phases. In the first phase, the system runs different experiments that involve different workflows for many diverse problems. The workflow may be generated automatically with the recourse to a given ontology of operators. The individual problems are characterized and the performance of different workflows recorded. This can be compared to running experiments with a set of classification algorithms and gathering the metaknowledge. In the second phase, the system carries out planning with the aim of designing a workflow that is likely to achieve good results. In this phase, a given ontology of operators can again be exploited. The expansion of the operators may be guided by the existing metaknowledge.

The aim is to give preference to the more promising expansions and generate a ranked list of viable workflows.

Cross-References

▶ Inductive Transfer

Recommended Reading

Bernstein A, Provost F, Hill S (2005) Toward intelligent assistance for a data mining process: an ontology-based approach for cost-sensitive classification. IEEE Trans Knowl Data Eng 17(4): 503–518

Brazdil P, Henery R (1994) Analysis of results. In: Michie D, Spiegelhalter DJ, Taylor CC (eds) Machine learning, neural and statistical classification. Ellis Horwood, New York

Brazdil P, Giraud-Carrier C, Soares C, Vilalta R (2009) Metalearning – applications to data mining. Springer, Berlin

Engels R, Theusinger C (1998) Using a data metric for offering preprocessing advice in data-mining applications. In: Proceedings of the 13th European conference on artificial intelligence, Brighton, pp 430–434

Hilario M, Nguyen P, Do H, Woznica A, Kalousis A (2011) Ontology-based meta-mining of knowledge discovery workflows. In: Jankowski N et al (eds) Meta-learning in computational intelligence. Springer, Berlin/New York

Kietz JU, Serban F, Bernstein A, Fischer S (2012) Designing KDD-workflows via HTN-planning for intelligent discovery assistance. In: Vanschoren J et al (eds) Planning to learn workshop at ECAI-2012 (PlanLearn-2012)

Leite R, Brazdil P, Vanschoren J (2012) Selecting classification algorithms with active testing. In: Machine learning and data mining in pattern recognition. Springer, Berlin/New York, pp 117–131

Mitchell T (1997) Machine learning. McGraw Hill, New York

Nakhaeizadeh G, Schnabl A (1997) Development of multi-criteria metrics for evaluation of data mining algorithms. In: Proceedings of the 3rd international conference on knowledge discovery and data mining, Newport Beach, pp 37–42

Pfahringer B, Bensusan H, Giraud-Carrier C (2000) Meta-learning by landmarking various learning algorithms. In: Proceedings of the 17th international conference on machine learning, Stanford, pp 743–750

Rice JR (1976) The algorithm selection problem. Adv Comput 15:65–118

Smith-Miles KA (2008) Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput Surv 41(1):6

Sun Q, Pfahringer B (2012) Bagging ensemble selection for regression. In: Proceedings of the 25th Australasian joint conference on artificial intelligence, Sydney, pp 695–706

Sun Q, Pfahringer B (2013) Pairwise meta-rules for better meta-learning-based algorithm ranking. Mach Learn 93(1):141–161

Vilalta R, Drissi Y (2002) A perspective view and survey of metalearning. Artif Intell Rev 18(2): 77–95

Xu L, Hutter F, Hoos H, Leyton-Brown K (2008) Cross-disciplinary perspectives on meta-learning for algorithm selection. J Artif Intell Res 32: 565–606

Minimum Cuts

▶ Graph Clustering

Minimum Description Length Principle

Teemu Roos
Department of Computer Science, Helsinki Institute for Information Technology, University of Helsinki, Helsinki, Finland

Abstract

The minimum description length (MDL) principle states that one should prefer the model that yields the shortest description of the data when the complexity of the model itself is also accounted for. MDL provides a versatile approach to statistical modeling. It is applicable to model selection and regularization. Modern versions of MDL lead to robust methods that are well suited for choosing an appropriate model complexity based on the data, thus extracting the maximum amount of information from the data without over-fitting. The modern

versions of MDL go well beyond the familiar $\frac{k}{2}\log n$ formula.

Philosophy

The MDL principle is a formal version of Occam's razor. While the Occam's razor only suggests that between hypotheses that are compatible with the evidence, one should choose the simplest one, the MDL principle also quantifies the compatibility of the hypotheses with the evidence. This leads to a trade-off between the complexity of the hypothesis and its compatibility with the evidence ("goodness of fit").

The philosophy of the MDL principle emphasizes that the evaluation of the merits of a model should not be based on its closeness to a "true" model, whose existence is often impossible to verify, but instead on the data. Inspired by Solomonoff's theory of universal induction, Rissanen postulated that a yardstick of the performance of a statistical model is the probability it assigns to the data. Since the probability is intimately related to code length (see below), the code length provides an equivalent way to measure performance. The key idea made possible by the coding interpretation is that the length of the description of the model itself can be quantified in the same units as the code length of the data, namely, bits. Earlier, Wallace and Boulton had made a similar proposal under the title minimum message length (MML) (Wallace and Boulton 1968). A fundamental difference between the two principles is that MML is a Bayesian approach while MDL is not.

The central tenet in MDL is that the better one is able to discover the regular features in the data, the shorter the code length. Showing that this is indeed the case often requires that we assume, for the sake of argument, that the data are generated by a true distribution and verify the statistical behavior of MDL-based methods under this assumption. Hence, the emphasis on the freedom from the assumption of a true model is more pertinent in the philosophy of MDL than in the technical analysis carried out in its theory.

Theory

The theory of MDL addresses two kinds of questions: (i) the first kind asks what is the shortest description achievable using a given model class, i.e., universal data compression; (ii) the second kind asks what can be said about the behavior of MDL methods when applied to model selection and other machine learning and data mining tasks. The latter kind of questions are closely related to the theory of statistical estimation and statistical learning theory. We review the theory related to these two kinds of questions separately.

Universal Data Compression

As is well known in information theory, the shortest expected code length achievable by a uniquely decodable code under a known data source, p^*, is given by the entropy of the source, $H(p^*)$. The lower bound is achieved by using a code word of length $\ell^*(x) = -\log p^*(x)$ bits for each source symbol x. (Here and in the following, log denotes base-2 logarithm.) Correspondingly, a code-length function ℓ is optimal under a source distribution defined by $q(x) = 2^{-\ell(x)}$. (For the sake of notational simplicity, we omit a normalizing factor $C = \sum_x 2^{-\ell(x)}$ which is necessary in case the code is not complete. Likewise, as is customary in MDL, we ignore the requirement that code lengths be integers.) These results can be extended to data sequences whereupon we write $x^n = x_1 \dots x_n$ to denote a sequence of length n.

While the case where the source distribution p^* is known can be considered solved in the sense that the average-case optimal code-length function ℓ^* is easily established as described above, the case where p^* is unknown is more intricate. Universal data compression studies similar lower bounds when the source distribution is not known or when the goal is not to minimize the expected code length. For example, when the source distribution is only known to be in a given *model class* (a set of distributions), $\mathcal{M}$, the goal may be to find a code that minimizes the *worst-case* expected code length under any source distribution $p^* \in \mathcal{M}$. A uniquely decodable code that achieves near-optimal code lengths with respect to a given model class is said to be *universal*.

Rissanen's groundbreaking 1978 paper (Rissanen 1978) gives a general construction for universal codes based on *two-part codes*. A two-part code first includes a code for encoding a distribution, q, over source sequences. The second part encodes the data using a code based on q. The length of the second part is thus $-\log q(x^n)$ bits. The length of the first part, $\ell(q)$, depends on the complexity of the distribution q, which leads to a trade-off between complexity measured by $\ell(q)$ and goodness of fit measured by $\log q(x)$:

$$\min_q(\ell(q) - \log q(x^n)). \tag{1}$$

For parametric models that are defined by a continuous parameter vector θ, a two-part coding approach requires that the parameters be quantized so that their code length is finite. Rissanen showed that given a k-dimensional parametric model class, $\mathcal{M} = \{p_\theta\,;\, \theta \in \Theta \subset \mathbb{R}^k\}$, the optimal quantization of the parameter space Θ is achieved by using accuracy of order $1/\sqrt{n}$ for each coordinate, where n is the sample size. The resulting total code length behaves as $-\log \hat{p}(x^n) + \frac{k}{2}\log n + \mathcal{O}(1)$, where $\hat{p}(x^n) = \max\{p_\theta(x^n) : \theta \in \Theta\}$ is the maximum probability under model class $\mathcal{M}$. Note that the leading terms of the formula are equivalent to the Bayesian information criterion (BIC) by Schwarz (Schwarz 1978). Later, Rissanen also showed that this is a *lower bound* on the code length of any universal code that holds for all but a measure-zero subset of sources in the given model class (Rissanen 1986).

The above results have subsequently been refined by studying the asymptotic and finite-sample values of the $\mathcal{O}(1)$ residual term for specific model classes. The resulting formulas lead to a more accurate characterization of model complexity, often involving the Fisher information (Rissanen 1996).

Subsequently, Rissanen and others have proposed other kinds of universal codes that are superior to two-part codes. These include Bayes-type mixture codes that involve a prior distribution for the unknown parameters (Rissanen 1986), predictive forms of MDL (Rissanen 1984; Wei 1992), and, most importantly, normalized maximum likelihood (NML) codes (Yuri 1987; Rissanen 1996). The latter have the important point-wise minimax property that they achieve the minimum worst-case point-wise redundancy:

$$\min_q \max_{x^n} -\log q(x^n) + \log \hat{p}(x^n),$$

where the maximum is over all possible data sequences of length n and the minimum is over all distributions.

Behavior of MDL-Based Learning Methods

The philosophy of MDL suggests that data compression is a measure of the success in discovering regularities in the data, and hence, better compression implies better modeling. Showing that this is indeed the case is the second kind of theory related to MDL.

Barron and Cover proposed the *index of resolvability* as a measure of the hardness of estimating a probabilistic source in a two-part coding setting (see above) (Barron and Cover 1991). It is defined as

$$R_n(p^*) = \min_q \left(\frac{\ell(q)}{n} + D(p^* \,||\, q) \right),$$

where p^* is the source distribution and $D(p^* \,||\, q)$ denotes the Kullback-Leibler divergence between p^* and q. Intuitively, a source is easily estimable if there exists a simple distribution that is close to the source. The result by Barron and Cover bounds the Hellinger distance between the true source distribution and the distribution $\hat{q}$ minimizing the two-part code length, Eq. (1), as

$$d_H^2(p^*, \hat{q}) \le \mathcal{O}(R_n(p^*)) \quad \text{in } p^*\text{-probability}.$$

For model selection problems, consistency is often defined in relation to a fixed set of alternative model classes and a criterion that selects one of them given the data. If the criterion leads to the simplest model class that contains the true source distribution, the criterion is said to be consistent. (Note that the additional requirement that the selected model class is the simplest one is needed in order to circumvent a trivial solution in nested model classes where simpler models are subsets of more complex model classes.) There are a large number of results showing that various MDL-based model selection criteria are consistent; for examples, see the next section.

Applications

MDL has been applied in a wide range of applications. It is well suited for model selection problems where one needs not only to estimate continuous parameters but also their number and, more generally, the *model structure*, based on statistical data. Other approaches applicable in many such scenarios include Bayesian methods (including minimum message length), cross validation, and structural risk minimization (see Cross-References below).

Some example applications include the following:

1. Autoregressive models, Markov chains, and their generalizations such as *tree machines* were among the first model classes studied in the MDL literature, see Rissanen (1978, 1984) and Weinberger et al. (1995).
2. Linear regression. Selecting a subset of relevant covariates is a classical example of a situation involving models of variable complexity, see Speed and Yu (1993), Wei (1992), and Rissanen (2000).
3. Discretization of continuous covariates enables the use of learning methods that use discrete data. The granularity of the discretization can be determined by applying MDL, see Fayyad and Irani (1993).
4. The structure of probabilistic graphical models encodes conditional independencies and determines the complexity of the model. Their structure can be learned by MDL, see, e.g., Lam and Bacchus (1994) and Silander et al. (2010)

Future Directions

The development of efficient and computationally tractable codes for practically relevant model classes is required in order to apply MDL more commonly in modern statistical applications. The following are among the most important future directions:

- While the original $\frac{k}{2}\log n$ formula is still regularly referred to as "the MDL principle," future work should focus on modern formulations involving more advanced codes such as the NML and its variations.
- There is strong empirical evidence suggesting that coding strategies with strong minimax properties lead to robust model selection methods, see, e.g., Silander et al. (2010). Tools akin to the index of resolvability are needed to gain better theoretical understanding of the properties of modern MDL methods.
- Scaling up to modern big data applications, where model complexity regularization is crucial, requires approximate versions of MDL with sublinear computational and storage requirements. Predictive MDL is a promising approach in handling high-throughput streaming data scenarios.

Cross-References

- ▶ Cross-Validation
- ▶ Inductive Inference
- ▶ Learning Graphical Models
- ▶ Minimum Message Length
- ▶ Model Evaluation
- ▶ Occam's Razor
- ▶ Overfitting
- ▶ Regularization
- ▶ Structural Risk Minimization
- ▶ Universal Learning Theory

Recommended Reading

Good review articles on MDL include Barron et al. (1998); Hansen and Yu (2001). The textbook by Grünwald (2007) is a comprehensive and detailed reference covering developments until 2007 Grünwald (2007).

Barron A, Cover T (1991) Minimum complexity density estimation. IEEE Trans Inf Theory 37(4):1034–1054

Barron A, Rissanen J, Yu B (1998) The minimum description length principle in coding and modeling. IEEE Trans Inf Theory 44:2734–2760

Fayyad U, Irani K (1993) Multi-interval discretization of continuous-valued attributes for classification learning. In: Bajczy R (ed) Proceedings of the 13th International Joint Conference on Artificial Intelligence and Minimum Description Length Principle, Chambery. Morgan Kauffman

Grünwald P (2007) The Minimum Description Length Principle. MIT Press, Cambridge

Hansen M, Yu B (2001) Model selection and the principle of minimum description length. J Am Stat Assoc 96(454):746–774

Lam W, Bacchus F (1994) Learning Bayesian belief networks: an approach based on the MDL principle. Comput Intell 10:269–293

Rissanen J (1978) Modeling by shortest data description. Automatica 14(5):465–658

Rissanen J (1984) Universal coding, information, prediction, and estimation. IEEE Trans Inf Theory 30:629–636

Rissanen J (1986) Stochastic complexity and modeling. Ann Stat 14(3):1080–1100

Rissanen J (1996) Fisher information and stochasic complexity. IEEE Trans Inf Theory 42(1): 40–47

Rissanen J (2000) MDL denoising. IEEE Trans Inf Theory 46(7):2537–2543

Schwarz G (1978) Estimating the dimension of a model. Ann Stat 6(2):461–464

Silander T, Roos T, Myllymäki P (2010) Learning locally minimax optimal Bayesian networks. Int J Approx Reason 51(5):544–557

Speed T, Yu B (1993) Model selection and prediction: normal regression. Ann Inst Stat Math 45(1): 35–54

Wallace C, Boulton D (1968) An information measure for classification. Comput J 11(2):185–194

Wei C (1992) On predictive least squares principles. Ann Stat 20(1):1–42

Weinberger M, Rissanen J, Feder M (1995) A universal finite memory source. IEEE Trans Inf Theory 41(3):643–652

Yuri Shtarkov (1987) Universal sequential coding of single messages. Probl Inf Transm 23(3): 3–17

Minimum Message Length

Rohan A. Baxter
Australian Taxation Office, Sydney, NSW, Australia

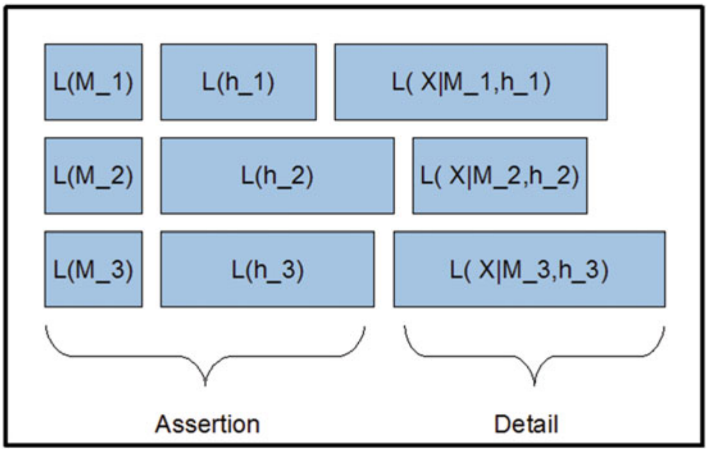
Abstract

The Minimum Message Length (MML) Principle is an information-theoretic approach to induction, hypothesis testing, model selection, and statistical inference. MML, which provides a formal specification for the implementation of Occam's Razor, asserts that the 'best' explanation of observed data is the shortest. MML was first published by Chris Wallace and David Boulton in 1968.

Definition

Minimum message length is a theory of ▸ inductive inference whereby the preferred model is the one minimizing the expected message length required to explain the data with the prior information.

Given the data represented in a finite binary string, E, an ▸ explanation of the data is a two-part ▸ message or binary string encoding the data to be sent between a sender and receiver. The first part of the message (the ▸ assertion) states a hypothesis, model, or theory about the source of the data. The second part (the ▸ detail) states those aspects of E which cannot be deduced from this assertion and prior knowledge. The sender and receiver are assumed to have agreed on the prior knowledge, the assertion code, and the detail code before the message is constructed and sent. The shared prior knowledge captures their belief about the data prior to seeing the data and is needed to provide probabilities or, equivalently, optimum codes, for the set of models. The assertion and detail codes can be equivalently considered to be the shared language for describing models (for the assertion code) and for describing data (for the detail code).



Minimum Message Length, Fig. 1 A view of model selection by MML. The data is coded assuming a model and parameters in the assertion. The model and parameters are coded in the assertion. As shown here, often different models have same probability, while the code lengths for model parameters and data detail differ between the different models

Out of all possible models which might be advanced about the data, MML considers the best inference as that model which leads to the shortest explanation. The length of the explanation can be calculated using ▶ Shannon's information, $L(E) = -\log(P(E))$, where $L(E)$ is the length of the shortest string encoding an event, E, and $P()$ is the probability of a message containing E.

To compare models, we calculate the explanation length for each and prefer the one with shortest explanation length. Figure 1 shows three models being evaluated and the different lengths of the assertion and details for each. Model 2 is preferred as it has the minimum message length.

Motivation and Background

The original motivation for minimum message length inductive inference is the idea that the best explanation of the facts is the shortest (Wallace and Boulton 1968). By inductive inference, we mean the selection of a best model of truth. This goal is distinct from a best model for prediction of future data or for choosing a model for making the most beneficial decisions. In the field of machine learning, greater focus has been on models for prediction and decision, but inference of the best models of truth has an important separate application.

For discrete models, MML looks like Bayesian model selection since choosing H to minimize the explanation length of data X

$$\begin{aligned} -\log P(H) - \log P(X|H) \\ = -\log(P(H)P(X|H)) \end{aligned}$$

is often, but not always, as discussed below, equivalent to choosing H to maximize the probability

$$P(H|X) \quad :$$

$$P(H|X) = \frac{P(H)P(X|H)}{P(X)}$$

where $P(X)$ is a constant for a given detail code.

For models with real-valued parameters, the equivalence between MML and Bayesian model selection always breaks down (Wallace 2005, p. 117). Stating the $P(H)$ in a message requires real-valued parameters in H to be stated to a limited precision. The MML coding approach replaces a continuum of possible hypotheses with a discrete subset of values and assigns a nonzero prior probability to each discrete theory. The discrete subsets are chosen to optimize the expected message length given the prior knowledge assumptions.

For models with only discrete-valued parameters, the equivalence between MML and Bayesian model selection may break down if the discrete values chosen involve the merging of values in the assumed prior distribution, $P(H)$ (Wallace 2005, p. 156). This may occur with a small dataset if the data is insufficient to justify a codebook distinguishing individual members of H.

Other than a discretized hypothesis space, MML shares many properties of Bayesian learning such as sufficiency, avoidance of overfitting, and consistency (Wallace 2005). One difference arising from the discretized hypothesis space is that MML allows inductive inference to be invariant under arbitrary monotonic transformations of parameter spaces. The Bayesian learning options for model choice such as the maximum a posteriori (MAP) estimate are not invariant under such transformations. Other theoretical benefits include consistency and guarantees against overfitting.

Message lengths of an explanation can be based on the theory of algorithmic complexity (Wallace and Dowe 1999), instead of Shannon's information. The algorithmic complexity (AC) of a string with respect to a universal Turing machine, T, can be related to Shannon's information by regarding T as defining a probability distribution over binary strings, $P(S)$, such that

$$P_{T(S)} = 2^{-AC(S)}, \forall S$$

The connection with algorithmic complexity has some appeal for applications involving data that are not random in a probabilistic sense, such as function approximation where data seems to be from a deterministic source. In these cases, after fitting a model, the data residuals can be encoded using AC randomness, since the probabilistic sense of randomness does not apply (Wallace 2005, p. 275).

Theory

Strict MML (SMML) estimators refer to the estimator functions which exactly minimize the expected message length (Wallace and Boulton 1975). Most practical MML estimators are not strict and are discussed in a separate section on Approximations.

A SMML estimator requires (Dowe et al. 2007):

- X, a data space, and a set of observations from the data space, $\{x_i : i \in N\}$.
- $p(x|h)$, a conditional probability function over data given a model, h.
- H is a model space. For example, H can be a simple continuum of known dimension k.
- $P(h)$: a prior probability density on the parameter space H: $\int_H P(h)dh = 1$.

X, H, and the functions P(h), $p(x|h)$ are assumed to be known *a priori* by both the sender and receiver of the explanation message. Both the sender and receiver agree on a code for X, using knowledge of X, H, $p(h)$, and $f(x|h)$ only.

The marginal prior probability of the data x follows from the assumed background knowledge:

$$r(x) = \int_H p(x|h)P(h)dh$$

The SMML estimator is a function $m : X \to H : m(x) = h$ which names the model to be selected.

The assertion, being a finite string, can name at most a countable subset of H. Call the subset $H^* = \{h_j : j = 1, 2, 3, \ldots\}$. The choice of H^* implies a coding distribution over $H^* : f(h_j) = q_j > 0 : j = 1, 2, 3, \ldots$ with $\sum_j q_j = 1$. So choice of $H^*$and q_j lead to a message length:

$$-\log q_j - \log p(x|h_j)$$

The sender, given x, will choose an h to make the explanation short. This choice is described by an estimator function: $m(x) : X \to H$ so that the length of the explanation is

$$I_1(x) = -\log q(m(x)) - \log p(x|m(x))$$

and the expected length is (Wallace 2005, p. 155):

$$I_1 = -\sum_{x \in X} r(x) [\log q(m(x)) \\ + \log p(x_i|m(x_i))]$$

Consider how to choose H^* and coding distribution q_j to minimize I_1. This will give the shortest explanation on average, prior to the sender seeing the actual data.

Define $t_j = \{x : m(x) = h_j\}$, so that t_j is the set of data which results in assertion h_j being used in the explanation. I_1 can now be written as two terms:

$$I_1 = -\sum_{h_j \in H_{start}} \left(\sum_{x_i \in t_j} r_i \right) \log q_j - \sum_{h_j \in H_{start}} \sum_{x_i \in t_j} r_i \log p(x_i|h_j)$$

The first term of $I_1$is minimized by choosing:

$$q_j = \sum_{x_i \in t_j} r_j$$

So the coding probability assigned to estimate $h_j$is the sum of the marginal probabilities of the data values resulting in h_j. It is the probability that estimate h_j will be used in the explanation based on the assumptions made.

The second term of I_1 is the average of the log likelihood over the data values used in h_j. Dowty (2013) gives a method for calculating the SMML estimator for a one-dimensional exponential family of statistical models with a continuous sufficient statistic. Techniques from differential geometry may lead to extensions of this work to linear and logistic regression models. This computational approach does not answer outstanding questions about the existence or uniqueness of SMML estimates.

Example with Binomial Distribution

This section describes the SMML estimator for the binomial distribution. For this problem with *100* independent trials giving success or failure, we have $p(x|p) = p^n(1-p)^{100} - s, h(p) = 1$, where s is the observed number of successes and p is the unknown probability of success.

We have a SMML estimator minimizing I_1 in Table 1. I_1 has *52.068* nits. Note that the partition p_j in Table 1 is not unique due to asymmetry in having *ten* partitions of *101* success counts. Note the difference between the SMML estimate, p_j, and the MAP estimate $\frac{s}{100}$ in this case. For example, for *50* observed successes, the MAP estimate is *0.5*, while SMML estimate is *0.58*. With *49* successes, the SMML estimate jumps to *0.41*, so it is very discrete. The SMML estimate spacings are consistent with the expected error and so the MAP estimates are arguably overly precise and smooth.

Minimum Message Length, Table 1 A SMML estimator for binomial distribution (Wallace 2005; Farr and Wallace 2002, p. 159)

j	s	*p_j*
1	0	0
2	1–6	0.035
3	7–17	0.12
4	18–32	0.25
5	33–49	0.41
6	50–66	0.58
7	67–81	0.74
8	82–93	0.875
9	94–99	0.965
10	100	1

This is less than *0.2* nits more than the optimal one-part code based on the marginal probability of the data $-\log r(x)$.

Approximations

SMML estimators are hard to find in practice and various approximations of SMML estimators have been suggested. We focus on the quadratic approximation here, often called the MML estimator or MML87 (Wallace and Freeman 1987). Other useful approximations have been developed and are described in Wallace (2005). The use of approximations in applications requires careful checking of the assumptions made by the approximation (such as various regularity conditions) to ensure the desirable theoretical properties of MML inductive inference still apply:

$$I_1(x) \approx -\log \frac{f(h')}{\sqrt{\frac{F(h')}{12}}} + \left[-\log p(x|h')\right] + \frac{0.5F(h',x)}{F(h')}$$

where $F(h)$ is the Fisher information:

$$F(h') = -E\frac{\partial^2}{(\partial h')^2}\log p(x|h')$$

$$= -\sum\nolimits_{x\in X} p(x|h')\frac{\partial^2}{(\partial h')^2}\log p(x|h')$$

The assumptions are (Wallace and Freeman 1987; Wallace 2005):

- *f*$(x|h)$ is approximately quadratic on theta near its maximum.
- H has a locally Euclidean metric.
- Fisher information is defined everywhere in H.
- $f(h)$ and $F(h)$ vary little over theta of order $\frac{1}{\sqrt{F(h)}}$.

A further approximation has the third term simplify to *0.5* only (Wallace 2005, p. 226) which assumes $F(h,x) \approx F(h)$.

The MML estimator is a discretized MAP estimator with the prior $P(h)$ being discretized as:

$$f(h') \approx \frac{P(h')}{\sqrt{F(h')}}$$

In practice, note that the Fisher information may be difficult to evaluate. Various approximations have been made for the Fisher information where appropriate for particular applications.

Applications

MML estimators have been developed for various probability distributions such as binomial, multinomial, and Poisson. MML estimators were developed for densities such as normal, von Mises, and Student's T (Wallace 2005). These estimators and associated coding schemes are then useful components for addressing more complex model selection problems in machine learning.

There have been many applications of MML estimators to model spaces from machine learning (Wallace 2005; O'Donnell et al. 2006; Allison 2009). We will now briefly note MML applications for mixture models, regular grammars, decision trees, and causal nets. MML estimators have also been developed for multiple linear regression (Wallace 2005), polynomial regression (Wallace 2005), neural networks (Allison 2009), ARMA time series, hidden Markov models (Edgoose and Allison 1999), sequence alignment (Allison 2009), phylogenetic trees (Allison 2009), factor analysis (Wallace 2005), cut-point estimation (Wallace 2005), and image segmentation.

Model-Based Clustering or Mixture Models

Clustering was the first MML application from Wallace and Boulton's 1968 paper (Wallace and Boulton 1968). Some changes to the coding scheme have occurred over the decades. A key development was the switch from definite assignment of classes to things to probabilistic assignment in the 1980s. The MML model selection and a particularly efficient search involving dynamic splitting and merging of clusters were implemented in a FORTRAN program called Snob (since it discriminated between things).

The assertion code consists of:

1. The number of classes
2. For each class
   2.1 The population proportion
   2.2 Parameters of the statistical distribution for each attribute (or an insignificant flag)

The detail code consists of, for each datum, the class to which it belongs, attribute values assuming the distribution parameters of the class. Bits-back coding is used to partially or probabilistically assign a class to each datum. This efficiency is needed to get consistent estimates.

Probabilistic Finite-State Machines

Probabilistic finite-state machines can represent probabilistic regular grammars (Wallace 2005). A simple assertion code for the discrete FSM structure, as developed by Wallace and Georgeff, is the following:

- State the number of states, S, using a prior $P(S)$.
- For each state, code the number of arcs leaving the state, $\log(K+1)$, where $K+1$ is maximum number of arcs possible.
- Code the symbols labeling the arcs, $\log\begin{pmatrix} K+1 \\ C \\ a_s \end{pmatrix}$.
- For each arc, code the destination state, $a_s \log S$.

The number of all states other than state 1 is arbitrary, so the code permits $(S-1)!$, equal length, and different descriptions of the same FSM. This inefficiency can be adjusted for by subtracting $\log(S-1)!$

A candidate detail code used to code the sentences is an incremental code where each transition from state to state is coded incrementally, using $\log n_{sk} + \frac{1}{v_s} + a_s$ where n_{sk} is the number of times this arc has already been followed and v_s is the number of times the state has already been left.

This application illustrates some general issues about assertion codes for discrete structures:

1. There can be choices about what to include in the assertion code or not. For example, the transition probabilities are not part of the assertion code above, but could be included, with adjustments, in an alternative design (Wallace 2005).
2. Simple approaches with interpretable priors may be desirable even if using non-optimal codes. The assumptions made should be validated. For example, arcs between states in FSMs are usually relatively sparse (a_s ≪ S) so a uniform distribution is not a sensible prior here.
3. Redundancy comes from being able to code equivalent models with different descriptions. For some model spaces, determining the equivalence is either not possible or very expensive computationally.
4. Redundancy can come from the code allowing description of models that cannot arise. For example, the example assertion code could describe a FSM with states with no arcs.
5. Exhaustive search of model space can only be done for small FSMs. For larger applications, the performance of the MML model selection is conflated with performance of the necessary search space heuristics. This issue also occurs with decision trees, causal nets, etc.

In a particular application, it may be appropriate to trade-off redundancy with interpretability in assertion code design.

Decision Trees

Assertion codes for decision trees and graphs have been developed (Wallace and Patrick 1993; Wallace 2005). An assertion describes the structure of the tree, while the detail code describes the target labels. The number of attributes, the arity of each attribute, an agreed attribute order, and probability that a node is a leaf or split node are assumed known by the sender and receiver. Like the PFSM transition probabilities, the leaf class distributions are not explicitly included in the decision tree model (a point of distinction from Bayesian tree approaches).

An assertion code can be constructed by performing a prefix traversal of the tree describing each node. Describing a node requires −log_2 P_L if it is a leaf and −log_2 P_s if it is a split node. If it is a split node, the attribute that it splits on must be specified, requiring log_2 (number of available attributes). If it is a leaf node, the data distribution model should be specified. For example, the parameters of a binomial distribution if the data consists of two classes.

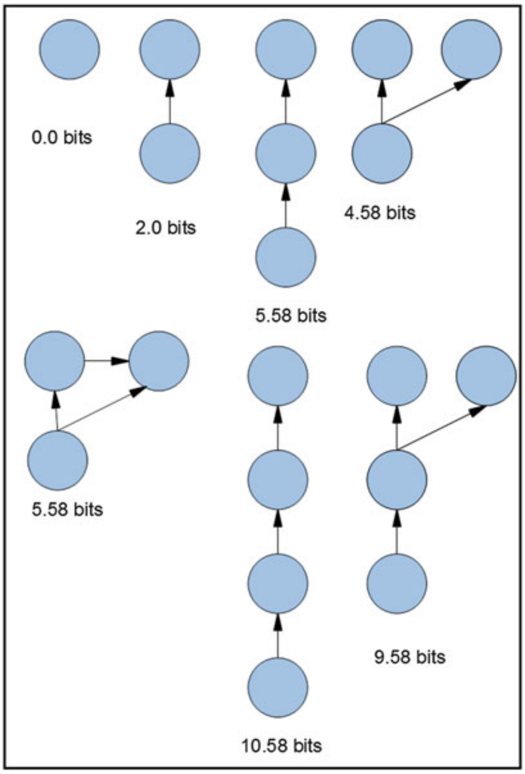
Causal Nets (Dai et al. 1997; Neil et al. 1999; O'Donnell et al. 2006)

The assertion code has two parts.
First part: DAG

1. Specify an ordering of variables, log $N!$.
2. Specify which of M_a possible arcs are present, $\log(N(N-1)/2)$ bits on assumption probability an arc is present is 0.5.

   Second part: Parameters
3. For each variable, state the form of conditional distribution and then the parameters of the distribution. Then encode all N values of v_j according to the distribution.

Note that the assertion code is intermixed with the detail code for each variable (Wallace 2005). Further adjustments are made to deal with grouping of causal nets with various equivalences or near equivalences. This requires a further approximation because no attempt is made to code the best representative causal net from the group of causal nets described (Fig. 2).



Minimum Message Length, Fig. 2 Assertion code lengths for different DAGS using the example coding scheme

Future Directions

There seems a potential for further development of feasible approximations that maintain the key SMML properties. The crossover of exciting new developments in coding theory may also help with the development of MML estimators. Examples include stochastic encoding such as bits-back coding, discovered by Wallace in 1990 (Wallace 1990) and since expanded to many new application areas showing connections between MML with variational learning and ensemble learning (Honkela and Valpola 2004). Another area is the relationship between optimum hypothesis discretization and indices of resolvability and rate-distortion optimization (Lanterman 2001).

MML estimators will continue to be developed for the new model spaces that arise in machine learning. MML relevance seems assured because with complex models, such as social networks, the best model is the useful outcome, rather than a prediction or posterior distribution of networks.

Open-source software using MML estimators for difference machine learning models is available (MML software).

Definition of Key Terms Used Above

Inductive inference: Choice of a model, theory, or hypothesis to express an apparent regularity or pattern in a body of data about many particular instances or events.

Explanation: A code with two parts, where the first part is an assertion code and the second part is a detail code.

Assertion: The code or language shared between the sender and receiver that is used to describe the model.

Detail: The code or language shared between the sender and receiver that is used to describe the data conditional on the asserted model.

Message: A binary sequence conveying information is called a message.

Shannon's information: If a message announces an event E_1 of probability $P(E_1)$, its information content is $-\log_2 P(E_1)$. This is also its length in bits.

Cross-References

- ▶ Bayesian Methods
- ▶ Inductive Inference
- ▶ Minimum Description Length Principle
- ▶ Universal Learning Theory

Recommended Reading

Allison L (2009) MML website. http://www.allisons.org/ll/MML/

Dowe DL, Gardner SB, Oppy G (2007) Bayes not bust!: why simplicity is no problem for Bayesians. Brit J Phil Sci 58:709–754

Dowty JG (2013) SMML estimators for 1-dimensional continuous data. Comput J. doi:10.1093/comjnl/bxt145

Dai H, Korb KB, Wallace CS, Wu X (1997) A study of causal discovery with weak links and small samples. In: Proceedings of fifteenth international joint conference on artificial intelligence. Morgan Kaufman, San Francisco, pp 1304–1309

Edgoose T, Allison L (1999) MML Markov classification of sequential data. Stat Comput 9(4):269–278

Farr GE, Wallace CS (2002) The complexity of strict minimum message length inference. Comput J 45(3): 285–292

Grunwald P (2008) The minimum description length principle. MIT Press, Cambridge

Honkela A, Valpola H (2004) Variational learning and bits-back coding: an information-theoretic view to Bayesian learning. IEEE Trans Neural Netw 15(4):800–810

Lanterman AD (2001) Schwarz, Wallace and Rissanen: intertwining themes in theories of model selection. Int Stat Rev 69(2):185–212

MML software: www.datamining.monash.edu.au/software, http://allisons.org/ll/Images/People/Wallace/FactorSnob/

Neil JR, Wallace CS, Korb KB, Learning Bayesian networks with restricted interactions, in Laskey and Prade. In: Proceedings of the fifteenth conference of uncertainty in artificial intelligence (UAI-99), Stockholm, pp 486–493

O'Donnell R, Allison L, Korb K (2006) Learning hybrid Bayesian networks by MML. Lecture notes in computer science: AI 2006 – Advances in artificial intelligence, vol 4304. Springer, Berlin/New York, pp 192–203

Wallace CS (1990) Classification by minimum-message length inference. In: Akl SG et al (eds) Advances in computing and information-ICCI 1990. No. 468 in lecture notes in computer science. Springer, Berlin

Wallace CS (2005) Statistical & inductive inference by MML. Information sciences and statistics. Springer, New York

Wallace CS, Boulton DM (1968) An information measure for classification. Comput J 11:185–194

Wallace CS, Boulton DM (1975) An information measure for single-link classification. Comput J 18(3):236–238

Wallace CS, Dowe DL (1999) Minimum message length and Kolmogorov complexity. Comput J 42(4):330–337

Wallace CS, Freeman PR (1987) Estimation and inference by compact coding. J. R. Stat. Soc. (Ser B) 49:240–252

Wallace CS, Patrick JD (1993) Coding decision trees. Mach Learn 11:7–22

Mining a Stream of Opinionated Documents

▶ Opinion Stream Mining

Missing Attribute Values

Ivan Bruha
McMaster University, Hamilton, ON, Canada

Synonyms

Missing values; Unknown attribute values; Unknown values

Definition

When inducing ▶ decision trees or ▶ decision rules from real-world data, many different aspects must be taken into account. One important aspect, in particular, is the processing of *missing* (*unknown*) ▶ attribute values. In machine learning (ML), instances (objects, observations) are usually represented by a list of attribute values; such a list commonly has a fixed length (i.e., a fixed number of attributes).

The topic of missing attribute values has been analyzed in the field of ML in many papers (Brazdil and Bruha 1992; Bruha and Franek 1996; Karmaker and Kwer 2005; Long and Zhang 2004; Quinlan 1986, 1989). Grzymala-Busse (2003) and Li and Cercone (2006) discuss the treatment of missing attribute values using the rough set strategies.

There are a few directions in which missing (unknown) attribute values as well as the corresponding routines for their processing may be studied and designed. First, the *source of "unknownness"* should be investigated; there are several such sources (Kononenko 1992):

- A value is *missing* because it was forgotten or lost
- A certain attribute is *not applicable* for a given instance (e.g., it does not exist for a given observation)
- An attribute value is *irrelevant* in a given context
- For a given observation, the designer of a training database does not care about the value of a certain attribute (the so-called *dont-care* value)

The first source may represent a random case, while the remaining ones are of structural character.

Moreover, it is important to define formulas for *matching instances* (examples) containing missing attribute values with decision trees and decision rules as different matching routines vary in this respect.

Strategies for Missing Value Processing

The aim of this section is to survey the well-known strategies for the processing of missing attribute values. Quinlan (1989) surveys and investigates quite a few techniques for processing unknown attribute values processing for the TDIDT family. This chapter first introduces the seven strategies that are applied in many ML algorithms. It then discusses particular strategies for the four paradigms: Top Down Induction Decision Trees (TDIDT), (also known as the decision tree paradigm, or divide-and-conquer), covering paradigm (also known as the decision rules paradigm), Naive Bayes, and induction of ▶ association rules. The conclusion compares the above strategies and then portrays possible directions in combining these strategies into a more robust system.

To deal with real-world situations, it is necessary to process incomplete data – i.e., data with missing (unknown) attribute values. Here we introduce the seven strategies (routines) for processing missing-attribute-values. They differ in the style of the solution of their matching formulae. There are the following natural ways of dealing with unknown attribute values:

1. Ignore the example (object, observation) with missing values: strategy *Ignore* (*I*)
2. Consider the missing (unknown) value as an additional regular value for a given attribute: strategy *Unknown* (*U*) or
3. Substitute the missing (unknown) value for matching purposes by a suitable value which is either
   - The most common value: strategy *Common* (*C*)
   - A proportional fraction: strategy *Fraction* (*F*)
   - Any value: strategy *Anyvalue* (*A*)
   - Random value: strategy *Random* (*Ran*)
   - A value determined by a ML approach: strategy *Meta-Fill-In* (*M*) of the known values of the attribute that occur in the training set

Dealing with missing attribute values is in fact determined by matching a selector (see the corresponding definitions below) with an instance. A matching procedure of a selector with a fully specified instance returns the uniform solution: the instance either matches or not. Dilemmas arise when a partially defined instance is to be matched.

We now informally introduce a couple of definitions. An inductive algorithm generates a knowledge base (decision tree or a set of decision rules) from a*training set* of K training examples, each accompanied by its desired ▶ class $C_r, r = 1, \dots, R$. Examples are formally represented by N ▶ attributes, which are either discrete (symbolic) or numerical (continuous). A discrete attribute $A_n, n = 1 \dots, N$, comprises $J(n)$ distinct values $V_1, \dots, V_{J(n)}$. A numerical attribute may attain any value from a continuous interval. The symbolic/logical ML algorithms usually process the numerical attributes by ▶ discretization/fuzzification procedures, either on-line or off-line; see e.g., Bruha and Berka (2000).

An example (object, observation) can thus be expressed as an N-tuple $\mathbf{x} = [x_1, \dots, x_N]$, involving N attribute values. A *selector* S_n is defined as an attribute-value pair of the form $x_n = V_j$, where V_j is the jth value of the attribute A_n (or the jth interval of a numerical attribute A_n).

To process missing values, we should know in advance (for $r = 1, \dots, R, n = 1, \dots, N, j = 1, \dots, J(n)$):

- The overall*absolute* frequencies $F_{n,j}$ that express the number of examples exhibiting the value V_j for each attribute A_n
- The *class-sensitive absolute* frequencies $F_{r,n,j}$ that express the number of examples of the class C_r exhibiting the value V_j for each attribute A_n
- The *overall relative* frequencies $f_{n,j}$ of all known values V_j for each attribute A_n
- The *class-sensitive relative* frequencies $f_{r,n,j}$ of all known values V_j for each attribute A_n and for a given class C_r

The underlying idea for learning relies on the class distribution; i.e., the class-sensitive frequencies (overall and class-sensitive frequencies) are utilized. As soon as we substitute a missing value by a suitable one, we take the desired class of the example into consideration in order not to increase the noise in the data set. On the other hand, the overall frequencies are applied within classification.

We can now define the matching of an example $\mathbf{x}$ with a selector S_n by the so-called *matching ratio* = 0 if $x_n \neq V_j$

$$\mu(\mathbf{x}, S_n)\{= 1 \text{ if } x_n = V_j\} \qquad (1)$$
$$\in [0; 1] \text{ if } x_n \text{ is unknown (missing)}$$

A particular value of the matching ratio is determined by the selected routine (strategy) for missing value processing.

(I) Strategy Ignore: Ignore Missing Values: This strategy simply ignores examples (instances) with at least one missing attribute value before learning. Hence, no dilemma arises when determining matching ratios within learning. However, this approach does not contribute to any enhancement of processing of noisy or partly specified data.

As for classification, a missing value does not match any regular (known) value of a selector. Thus, a selector's matching ratio is equal to 0 for any missing value. Consequently, only a path of nodes in a decision tree or a decision rule that tests only the regular values during classification may succeed. If there is no such path of nodes in a decision tree or such a rule has not been found, then the default principle is applied; i.e., the instance with missing value(s) is classified as belonging to the majority class.

(U) Strategy Unknown: Unknown Value as a Regular One: An unknown (missing) value is considered as an additional attribute value. Hence, the number of values is increased by one for each attribute that depicts an unknown value in the training set. The matching ratio of a selector comprising the test of the selector S_n and an instance with the nth attribute missing is equal to 1 if this test (selector) is of the form

x_n =? where "?," represents the missing (unknown) value.

Note that selectors corresponding to the numerical (continuous) attributes are formed by tests $x_n \in V_j$ (where V_j is a numerical interval) or x_n =?.

(C) Strategy Common: The Most Common Value: This routine needs the class-sensitive absolute frequencies $F_{r,n,j}$ to be known before the actual learning process, and the overall frequencies $F_{n,j}$ before the classification. A missing value of a discrete attribute A_n of an example belonging to the class C_r is replaced by the *class-sensitive common* value, which maximizes the Laplacian formula $\frac{F_{r,n,j}+1}{F_{n,j}+R}$ over j for the given r and n. If the maximum is reached for more than one value of A_n, then the value V_j with the greatest frequency $F_{r,n,j}$ is selected as the common value.

A missing value within the classification is replaced by the *overall common* value, which maximizes $F_{n,j}$ over the subscript j. Consequently, the matching ratio yields 0 or 1, as every missing value is substituted by a concrete, known value.

The Laplacian formula utilized within the learning phase prefers those attribute values that are more predictive for a given class, contrary to the conventional "maximum frequency" scheme. For instance, let an attribute have two values: the value V_1 with the absolute frequencies [4, 2] for the classes C_1 and C_2, and the value V_2 with frequencies [3, 0] for these two classes. Then, when looking for the most common value of this attribute for the class C_1, the maximum frequency chooses the value V_1 as the most common value, whereas the Laplacian formula prefers the value V_2 as the more predictive for the class C_1.

(F) Strategy Fraction: Split into Proportional Fractions:

- Learning phase

The learning phase requires that the relative frequencies $f_{r,n,j}$ above the entire training set be known. Each example $\mathbf{x}$ of class C_r with a missing value of a discrete attribute A_n is substituted by a collection of examples before the actual learning phase, as follows: the missing value of A_n is replaced by all known values V_j of A_n and C_r. The weight of each split example (with the value V_j) is

$$w_j = w(\mathbf{x}) * f_{r,n,j},\ j = 1, \ldots, J(n)$$

where $w(\mathbf{x})$ is the weight of the original example $\mathbf{x}$. The weight is assigned by the designer of the training set and represents the designer's subjective judgment of the importance of that particular example within the entire training set. The matching ratio of the split examples is accomplished by (1) in a standard way.

If a training example involves more missing attribute values, then the above splitting is done for each missing value. Thus, the matching ratio may rapidly decrease. Therefore, this strategy, *Fraction,* should involve a methodology to avoid explosion of examples, so that only a predefined number of split examples with the largest weights is used for replacement of the original example.

- Classification phase

The routine *Fraction* works for each paradigm in a different way. In case of a decision tree, the example with a missing value for a given attribute A_n is split along all branches, with the weights equal to the overall relative frequencies $f_{n,j}$.

As for the decision rules, the matching ratio for a selector $x_n = V_j$ is defined by (1) as $\mu = f_{n,j}$ for a missing value of A_n. An instance with a missing value is tested with the conditions of all the rules, and is attached to the rule whose condition yields the maximum matching ratio – i.e., it is assigned to the class of this rule.

(A) Strategy Anyvalue: Any Value Matches: A missing value matches any existing attribute value, both in learning and classification. Therefore, a matching ratio μ of any selector is equal to 1 for any missing value.

It should be noticed that there is no uniform scheme in machine learning for processing the "any-value." In some systems, an example with a missing value for attribute A_n is replaced by

$J(n)$ examples in which the missing value is in turn substituted by each regular value $V_j, j = 1, \ldots, J(n)$. In other systems, the missing "anyvalue" is substituted by any first attribute value involved in a newly generated rule when covered examples are being removed from the training set; see Bruha and Franek (1996) for details.

(Ran) Strategy Random: Substitute by Random Value A missing value of an attribute A_n is substituted by a randomly selected value from the set of its values $V_j, j = 1, \ldots, J(n)$. In case of the numerical attributes, the process used in the routine *Common* is first applied, i.e., the entire numerical range is partitioned into a prespecified number of equal-length intervals. A missing value of the numerical attribute is then substituted by the mean value of a randomly selected interval.

At least two possibilities exist in the random procedure. Either

- A value is randomly chosen according to the uniform distribution – i.e., all the values have the same chance
- A value is chosen in conformity with the value distribution – i.e., the most frequent value has the greatest chance of being selected

To illustrate the difference of the strategies Anyvalue and Random, consider this scheme. Let the attribute A have three possible values, V_1, V_2, V_3 with the relative distribution [0.5, 0.3, 0.2]. (Here, of course, we consider class-sensitive distribution for the learning phase, overall one for classification.)

Strategy Anyvalue for TDIDT replaces the missing value $A = ?$ by each possible value $A = V_j$, $j = 1, 2, 3$, and these selectors (attribute-value pairs) are utilized for selecting a new node (during learning), or pushed down along an existing decision tree (classi-fication).

Strategy Anyvalue for covering algorithms: if the corresponding selector in a complex is for example, $A = V_3$ then the selector $A = ?$ in an instance is replaced by $A = V_3$, so that the matching always succeeds.

Let the pseudo-random number be for example, 0.4 in the strategy Random. Then, in the first case – i.e., uniform distribution (one can consider the relative distribution has been changed to [0.33, 0.33, 0.33]) – the missing value $A = ?$ is replaced by $A = V_2$. In the second possibility – i.e., the actual distribution – the missing value is replaced by $A = V_1$.

(M) Strategy Meta Fill In: Use Another Learning Topology for Substitution: This interesting strategy utilizes another ML algorithm in order to fill in the missing attribute values. This second (or *meta*) learning algorithm uses the remaining attribute values of a given example (instance, observation) for determining (inducing) the missing value of the attribute A_n. There are several approaches to this strategy.

The first one was designed by Breiman; it uses a *surrogate split* in order to determine the missing attribute value. We can observe that a surrogate attribute has the highest correlation with the original one.

Quinlan (1989) was the first to introduce the meta-fill-in strategy; in fact, this method was proposed by A. Shapiro during their private communication. It builds a decision tree for each attribute that attempts to derive a value of the attribute with a missing value for a given instance in terms of the values of other attributes of the given instance.

Lakshminarayan et al. (1996) introduced a more robust approach where a ML technique (namely, C4.5) is used to fill in the missing values.

Ragel and Cremilleux (1998) developed a fill-in strategy by using the association rules paradigm. It induces a set of association rules according to the entire training set. This method is able to efficiently process the missing attribute values.

Missing Value Processing Techniques in Various ML Paradigms

As mentioned above, various missing value processing techniques have been embedded into various ML paradigms. We introduce four such systems.

Quinlan (1986, 1989) applied missing value techniques into ID3, the most famous TDIDT (decision tree inducing) algorithm. His list exhibits two additional routines that were not discussed above:

- The evaluation of an attribute uses the routines I, C, M, and R (i.e., reduce the apparent information gain from evaluating an attribute by the proportion of training examples with the missing value for this attribute)
- When partitioning a training set using the selected attribute, the routines I, U, C, F, A, M were used
- The classification phase utilizes the strategies U, C, F, M, and H (i.e., halt the classification and assign the instance to the most likely class)

Quinlan then combined the above routine into triples each representing a different overall strategy; however, not all the possible combinations of these routines make sense.

His experiments revealed that the strategies starting with R or C behave reasonably accurately among them the strategy *RFF* is the best. Brazdil and Bruha (1992) improved this strategy for partitioning a training set. They combined the strategies U and F; therefore, they call it *R(UF)(UF)* strategy.

Bruha and Franek (1996) discusses the embedding of missing value strategies into the covering algorithm CN4 (Bruha and Kockova 1994), a large extension of the well-known CN2 (Clark and Niblett 1989). A condition of a decision rule has the form:

$$\text{cmplx} = S_{q1} \& \ldots \& S_{qM}$$

where $S_{qm}, m = 1, \ldots, M$, is the mth selector testing the jth value V_j of the q_mth attribute, (i.e., exhibiting the form $x_{qm} = V_j$). For the purposes of processing missing values, we need to define the *matching ratio* of the example $\mathbf{x}$ and the rule's condition *Cond*. (Bruha and Franek 1996) uses two definitions:

The product of matching ratios of its selectors:

$$\mu(x, \text{cmplx}) = w(x) \prod_{m=1}^{M} \mu(x, S_{qm}) \qquad (2)$$

or their average:

$$\mu(x, \text{cmplx}) = \frac{w(x)}{M} \sum_{m=1}^{M} \mu(x, S_{qm}), \qquad (3)$$

where $w(\mathbf{x})$ is the weight of the example $\mathbf{x}$ (1 by default), and μ on the right-hand side is the selector's matching ratio (1).

The Naive Bayes algorithm can process missing attribute values in a very simple way, because the probabilities it works with are, in fact, the relative frequencies discussed above: the class-sensitive relative frequencies $f_{r,n,j}$ (for the learning phase) and the overall relative frequencies $f_{n,j}$ (for the purposes of classification). When learning relative frequencies, all strategies can by applied. Only routine Fraction is useless because it copies the distribution of the rest of a training set. When classifying an instance with missing value A_n =?, all strategies can be applied as well. Section Fraction substitutes this instances with $J(n)$ instances by each known attribute value, and each "fractioned" instance is attached by the weight $f_{n,j}$, and classified separately.

Ragel and Cremilleux (1998) present the missing value processing strategy for the algorithm that induced ▸ association rules. Their algorithm uses a modified version of the routine *Ignore*. The instances with missing attribute values are not removed from the training database but the missing values are ignored (or "hidden").

The experiments with the above techniques for handling missing values have revealed the following. In both decision tree and decision rules inducing algorithms, the routine *Ignore* is evidently the worst strategy. An Interesting issue is that the association rule inducing algorithms use its modified version. In case of the decision tree inducing algorithms, the strategy *Fraction* is one of the best; however, the decision rules inducing algorithms found it not so efficient. The explanation for this fact is based on different ways of processing examples in these two paradigms: in TDIDT, all training examples are eventually incorporated into the decision tree generated by the learning algorithm; on the other hand, the covering paradigm algorithm generates rules that may not cover all of the examples from the training set (as some of the examples are found not to be representable).

Although the routine *Unknown* is one of the "winners" (at least in the rule inducing algorithms and Brazdil and Bruha (1992), it is not quite clear how one can interpret, on a philosophical as well as a semantic level, a branch in a decision tree or a decision rule that involves a selector with an attribute equal to "?" (missing value). Strategy Fraction can be faced by "problems": if an example/instance exhibits too many missing values, then this strategy generates too many "fractioned" examples with very negligible weights.

One can find out that each dataset has more or less its own "favorite" routine for processing missing attribute values. It evidently depends on the magnitude of noise and the source of unknownness in each dataset. The problem of a "favorite" strategy can be solved by various approaches. One possibility is to create a small "window" within a training set, and to check the efficiency of each strategy in this window, and then choose the most efficient one. Bruha (2003) discusses another possibility: investigating the advantages of utilizing the external background (domain-specific, expert) knowledge on an attribute hierarchical tree.

Also, the concept of the so-called ▸ meta-combiner (Fan et al. 1996) can be utilized. A learning algorithm processes a given training base for each strategy for missing values independently; thus, all the missing value strategies are utilized in parallel and the meta-classifier makes up its decision from the results of the base level (Bruha 2004).

The above issue – i.e., selection or combination of various strategies for missing value processing – is an open field for future research.

Recommended Reading

Brazdil PB, Bruha I (1992) A note on processing missing attribute values: a modified technique. In: Workshop on machine learning, Canadian conference AI, Vancouver

Bruha I (2003) Unknown attribute value processing by domain-specific external expert knowledge. In: 7th WSEAS international conference on systems, Corfu

Bruha I (2004) Meta-learner for unknown attribute values processing: dealing with inconsistency of meta-databases. J Intell Inf Syst 22(1):71–84

Bruha I, Franek F (1996) Comparison of various routines for unknown attribute value processing: covering paradigm. Int J Pattern Recognit Artif Intell 10(8):939–955

Bruha I, Berka P (2000) Discretization and fuzzification of numerical attributes in attribute-based learning. In: Szczepaniak PS, Lisboa PJG, Kacprzyk J (eds) Fuzzy systems in medicine. Physica/Springer, Heidelberg/New York, pp 112–138

Bruha I, Kockova S (1994) A support for decision making: cost-sensitive learning system. Artif Intell Med 6: 67–82

Clark P, Niblett T (1989) The CN2 induction algorithm. Mach Learn 3:261–283

Fan DW, Chan PK, Stolfo SJ (1996) A comparative evaluation of combiner and stacked generalization. In: Workshop integrating multiple learning models, AAAI, Portland

Grzymala-Busse JW (2003) Rough set strategies to date with missing attribute values. In: IEEE conference on proceedings of workshop on foundations and new directions in data mining, data mining, pp 56–63

Karmaker A, Kwer S (2005) Incorporating an EM-approach for handling missing attribute-values in decision tree induction. In: International conference on hybrid intelligent systems, pp 6–11

Kononenko I (1992) Combining decisions of multiple rules. In: du Boulay B, Sgurev V (eds) Artificial intelligence V: methodology, systems, applications, pp 87–96. Elsevier

Lakshminarayan K et al (1996) Imputation of missing data using machine learning techniques. In: Conference knowledge discovery in databases (KDD-96), pp 140–145

Li J, Cercone N (2006) Assigning missing attribute values based on rough sets theory. In: IEEE international conference on granular computing, Atlanta, pp 31–37

Long WJ, Zhang WX (2004) A novel measure of compatibility and methods of missing attribute values treatment in decision tables. In: International conference on machine learning and cybernetics, pp 2356–2360

Quinlan JR (1986) Induction of decision trees. Mach Learn 1:81–106

Quinlan JR (1989) Unknown attribute values in ID3. In: Proceedings of international workshop on machine learning, pp 164–168

Ragel A, Cremilleux B (1998) Treatment of missing values for association rules. Lecture Notes in Computer Science, vol 1394, pp 258–270

Missing Values

▶ Missing Attribute Values

Mistake-Bounded Learning

▶ Online Learning

Mixture Distribution

▶ Mixture Model

Mixture Model

Rohan A. Baxter
Australian Taxation Office, Sydney, NSW,
Australia

Abstract

A mixture model is a probability model for representing subpopulations within a data set. The mixture model is built up from a weighted combination of component probability distributions. Mixture models can be estimated by attribution partial membership to the component distributions to individual observations in the data set.

Synonyms

Finite mixture model; Latent class model; Mixture distribution; Mixture modeling; Mixture modeling

Definition

A mixture model is a collection of probability distributions or densities $D_1, \dots, D_k$ and mixing weights or proportions $W_1, \dots, W_k$,where k is the number of component distributions (McLachlan and Peel 2000; Lindsey 1996; Duda et al. 2000).

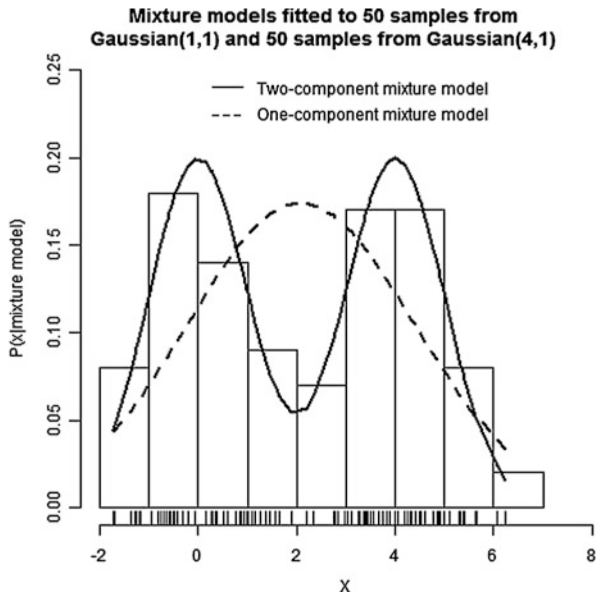
The mixture model, $P(x|D_1, \dots, D_k, w_1, \dots, w_k) = \sum_{j=1}^{k} w_j P(x|D_j)$, is a probability distribution over the data conditional on the component distributions of the mixture and their mixing weights. Mixture models can be used for density estimation, model-based clustering or unsupervised learning, and classification.

Figure 1 shows one-dimensional data plotted along the x-axis with tick marks and a histogram of that data. The probability densities of two mixture models fitted to that data are then shown. The one-component mixture model is a Gaussian density with mean around 2 and standard deviation of 2.3. The two-component mixture model has one component with mean around 0 and one component with mean around 4, which reflects how these simple example data was artificially generated. The two-component mixture model can be used for clustering by considering each of its components as a cluster and assigning cluster membership based on the relative probability of a data item belonging to that component. Data less than 2 will have higher probability of belonging to the Gaussian with mean 0 component.

Motivation and Background

Mixture models are easy and convenient to apply. They trade off good power in data representation with relative ease in building the models. When used in clustering, a mixture model will have a component distribution covering each cluster, while the mixing weights reflect the relative proportion of a cluster's population. For example, a two-component mixture model of seal skull lengths from two different seal species may have one component with relative proportion 0.3 and the other 0.7 reflecting the relative frequency of the two components.

Mixture Model, Fig. 1
Mixture model example for one-dimensional data



Estimation

In order to use mixture models, the following choices need to be made by the modeler or by the mixture model software, based on the characteristics of a particular problem domain and its datasets:

1. The type of the component distributions (e.g., Gaussian, multinomial, etc.)
2. The number of component distributions, k
3. The parameters for the component distributions (e.g., a one-dimensional Gaussian has a mean and standard deviation as parameters, and a higher dimensional Gaussian has a mean vector and covariance matrix as parameters)
4. Mixing weights, W_i
5. (Optional) component labels, c_j for each datum x_j, where $j = 1 \ldots n$ and n is the number of data

The fifth item above, component labels, are optional because they are only used in latent class mixture model frameworks where a definite component membership is part of the model specification. Other mixture model frameworks use probabilistic membership of each datum to each component distribution and so do not need explicit component labels.

The most common way of fitting distribution parameters and mixture weights is to use the expectation-maximization (EM) algorithm to find the maximum likelihood estimates. The EM algorithm is an iterative algorithm that, starting with initial guesses of parameter values, computes the mixing weights (the expectation step). The next step is to then compute the parameter values based on these weights (the maximization step). The expectation and maximization steps iterate and convergence is assured (Redner and Walker 2004). However there is no guarantee that a global optimum has been found, and so a number of random restarts may be required to find what other optima exist (Xu and Jordan 1996).

As an alternative to random restarts, a good search strategy can be to modify the current best solution, perhaps by choosing to split, merge, delete, or add component distributions at random. This can also be a way to explore mixture models with different number of components (Figueiredo and Jain 2002).

Since mixture models are a probabilistic model class, besides EM, other methods such as Bayesian methods or methods for graphical models can be used. These include Markov chain Monte Carlo inference and variational learning (Bishop 2006).

Choosing the Number of Components

The number of components in a mixture model is often unknown when used for clustering real-world data. There have been many methods for choosing the number of components. The global maximum for maximum likelihood chooses a component for every data item, which is usually undesirable. Criteria based on information theory or Bayesian model selection choose reasonable numbers of components in many domains (McLachlan and Peel 2000, Chap 6, 5). There is no universally accepted method, because there is no universally accepted optimality criteria for clustering or density estimation. The use of an infinite mixture model, by using an infinite number of components, is one way to avoid the number of component problem (Rasmussen 2000).

Types of Component Distributions

Besides Gaussian, other distributions can be used such as Poisson (for count data), von Mises (for data involving directions or angles), and Weibull. Heavy-tailed distributions require particular care because standard estimation may not work when mean or variance is infinite (Dasgupta et al. 2005).

Another commonly needed variation is a mixture model to handle a mix of continuous and categorical features (McLachlan and Peel 2000). For example, a binomial distribution can be used to model male/female gender proportions and Gaussian to model length for data relating to a seal species sample.

A further extension is to allow components to depend on covariates, leading to mixtures of regression models (McLachlan and Peel 2000). This leads to models such as mixtures of experts and hierarchical mixtures of experts (McLachlan and Peel 2000; Bishop 2006) which are flexible models for nonlinear regression. The combination of mixture models with hidden Markov models allows the modeling of dependent data (McLachlan and Peel 2000).

Large Datasets

The EM algorithm can be modified to find mixture models for very large datasets (Bradley et al. 2000). The modification allows for a single scan of the data and involves identifying compressible regions of the data.

Theory

A key issue for mixture models is learnability (Chaudri 2009). The more the component distributions overlap, the harder they are to learn. Higher dimensional data also makes learning harder. Sometimes, these problems can be overcome by increasing the data quantity, but, in extremely hard cases, this will not work (Xu and Jordan 1996; Srebo et al. 2006).

Another issue is the relationship between adequate sample size and the number of components. A pragmatic policy is to set minimum mixing weights for component distributions. For example, for a dataset of size 100, if mixing weights are required to be greater than 0.1, this implies a maximum of ten components that are possible to be learned from the data with these parameter settings.

Applications

Mixture model software is often available in the clustering or density estimation parts of general statistical and data mining software. More specialized mixture modeling software for clustering data has included Autoclass (Autoclass 2010), Snob (Snob 2010), and mclust (Mclust 2010).

Definition of Key Terms Used Above

Probability distribution: This is the probability for each value of a random variable with discrete values.

Probability density: This is a function of a continuous random variable that describes probability at a given point in the variable space.

Gaussian distribution: A bell-shaped probability density function with a peak at the mean. It has two parameters: the mean to give the location of the peak and the standard deviation to describe the width of the bell-shaped curve.

Mixing weights: These are the parameters of the mixture model giving the relative weights of each component distribution. The weights are between 0 and 1 and must sum to 1. In clustering applications, the mixing weights can be interpreted as the relative size of a cluster compared to the whole population.

Cross-References

- ▶ Density-Based Clustering
- ▶ Density Estimation
- ▶ Expectation Maximization Clustering
- ▶ Gaussian Distribution
- ▶ Graphical Models
- ▶ Learning Graphical Models
- ▶ Markov Chain Monte Carlo
- ▶ Model-based Clustering
- ▶ Unsupervised Learning

Recommended Reading

Autoclass (2010) http://ti.arc.nasa.gov/project/autoclass/. Last Accessed 22 Mar 2010

Bishop CM (2006) Pattern recognition and machine learning. Springer, New York

Bradley PS, Reina CA, Fayyad UM (2000) Clustering very large databases using EM mixture models. In: 15th international conference on pattern recognition, vol 2. Barcelona, pp 2076

Chaudri K (2010) Learning mixture models. http://themachinelearningforum.org/index.php/overviews/34-colt-overviews/53-learning-mixture-models.html. June 2009, Last Accessed 21 Mar 2010

Dasgupta A, Hopcroft J, Kleinberg J, Sandler M (2005) On learning mixtures of heavy-tailed distributions. In: Proceedings of foundations of computer science, Pittsburg

Duda RO, Hart PE, Stork DG (2000) Pattern classification, 2nd edn. Wiley-Interscience, New York

Figueiredo MAT, Jain AT (2002) Unsupervised learning of finite mixture models. IEEE Trans Pattern Anal Mach Intell 24:381–396

Lindsey BG (1996) Mixture models: theory, geometry and applications. IMS Publishers, Hayward

McLachlan GJ, Peel D (2000) Finite mixture models. Wiley, New York

Mclust (2010) http://www.stat.washington.edu/mclust/. Last Accessed 22 Mar 2010

Rasmussen CE (2000) The infinite Gaussian mixture model. In: NIPS 12. MIT Press, Cambridge, pp 554–560

Redner RA, Walker HF (2004) Mixture densities, maximum likelihood and the EM algorithm. SIAM Rev 26:195–239

Snob (2010) http://www.datamining.monash.edu.au/software/snob/. Last Accessed 22 Mar 2010

Srebo N, Shakhnarovich G, Roweis S (2006) An investigation of computational and informational limits in Gaussian mixture modeling. In: Proceedings of ICML, Pittsburgh

Xu L, Jordan MI (1996) On convergence properties of the EM algorithm for Gaussian mixtures. Neural Comput 8:129–151

Mixture Modeling

▶ Mixture Model

Mode Analysis

▶ Density-Based Clustering

Model Assessment

▶ Model Evaluation

Model Evaluation

Geoffrey I. Webb
Faculty of Information Technology, Monash
Clayton, Victoria, Australia

Abstract

Model evaluation is the process of assessing a property or properties of a model.

Synonyms

Evaluation of model performance; Model assessment; Assessment of model performance

Motivation and Background

It is often valuable to assess the efficacy of a model that has been learned. Such assessment is frequently relative – an evaluation of which of several alternative models is best suited to a specific application.

Processes and Techniques

There are many metrics by which a model may be assessed. The relative importance of each metric varies from application to application.

The primary considerations often relate to predictive efficacy – how useful will the predictions be in the particular context in which the model is to be deployed. Measures relating to predictive efficacy include ▶ accuracy, ▶ lift, ▶ mean absolute error, ▶ mean squared error, ▶ negative predictive value, ▶ positive predictive value, ▶ precision, ▶ recall, ▶ sensitivity, ▶ specificity, and various metrics based on ▶ ROC analysis.

Computational issues may also be important, such as a model's size or its execution time.

In many applications one of the most important considerations is the ease with which the model can be understood by the users or how consistent is it with the users' prior beliefs and understanding of the application domain.

When assessing the predictive efficacy of a model learned from data, to obtain a reliable estimate of its likely performance on new data, it is essential that it not be assessed by considering its performance on the data from which it was learned. A learning algorithm must interpolate appropriate predictions for regions of the ▶ instance space that are not included in the training data. It is probable that the inferred model will be more accurate for those regions represented in the training data than for those that are not, and hence predictions are likely to be less accurate for instances that were not included in the training data. Estimates that have been computed on the training data are called ▶ *resubstitution estimates*. For example, the error of a model on the training data from which it was learned is called resubstitution error.

Algorithm evaluation techniques such as ▶ cross-validation, ▶ holdout evaluation, and ▶ bootstrap sampling are designed to provide more reliable estimates of the accuracy of the models learned by an algorithm than would be obtained by assessing them on the training data.

Cross-References

▶ Algorithm Evaluation
▶ Overfitting
▶ ROC Analysis

Recommended Reading

Hastie T, Tibshirani R, Friedman J (2001) The elements of statistical learning. Springer, New York

Mitchell TM (1997) Machine learning. McGraw-Hill, New York

Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, Amsterdam/Boston

Model Selection

Model selection is the process of choosing an appropriate mathematical model from a class of models.

Model Space

▶ Hypothesis Space

Model Trees

Luís Torgo
University of Porto, Porto, Portugal

Synonyms

Functional trees, Linear regression trees, Piecewise linear models

Definition

Model trees are supervised learning methods that obtain a type of tree-based ▸ regression model, similar to ▸ regression trees, with the particularity of having functional models in the leaves instead of constants. These methods address multiple regression problems. In these problems we are usually given a training sample of n observations of a target continuous variable Y and of a vector of p predictor variables, $\mathbf{x} = X_1, \cdots, X_p$. Model trees provide an approximation of an unknown regression function $Y = f(\mathbf{x}) + \varepsilon$ with $Y \in \Re$ and $\varepsilon \approx N(0, \sigma^2)$. The leaves of these trees usually contain linear regression models, although some works also consider other types of models.

Motivation and Background

Model trees are motivated by the purpose of overcoming some of the known limitations of regression trees caused by their piecewise constant approximation. In effect, by using constants at the leaves, regression trees provide a coarse grained function approximation leading to poor accuracy in some domains. Model trees try to overcome this by using more complex models on the leaves. Trees with linear models in the leaves were first considered in Breiman and Meisel (1976) and Friedman (1979). Torgo (1997) has extended the notion of model trees to other type of models in the tree leaves, namely, kernel regression, later extended to other type of local regression models (Torgo 1999, 2000). The added complexity of the models used in the leaves increases the computational complexity of model trees when compared to regression trees and also decreases their interpretability. In this context, several works Chaudhuri et al. (1994), Dobra and Gehrke (2002), Loh (2002), Malerba et al. (2002), Natarajan and Pednault (2002), Torgo (2002), Malerba (2004), Potts and Sammut (2005), and Vogel et al. (2007) have focussed on obtaining model trees in a computationally efficient form.

Structure of Learning System

Approaches to model trees can be distinguished along two dimensions: the criterion used to select the best splits at each node, i.e., the criterion guiding the partitioning obtained by the tree, and the type of models used in the leaves. The choices along the first dimension are mainly driven by considerations of computational efficiency. In effect, the selection of the best split node involves evaluating many candidate splits. The evaluation of a binary split (the most common splits in tree-based models) consists in calculating the error reduction produced by the split, i.e.,

$$\Delta(s,t) = Err(t) - \left(\frac{n_{t_L}}{n_t} \times Err(t_L) + \frac{n_{t_R}}{n_t} \times Err(t_R) \right) \quad (1)$$

where t is a tree node with sub-nodes t_L and t_R originated by the split test s, while n_t, n_{t_L}, and n_{t_R} are the cardinalities of the respective sets of observations on each of these nodes, and $Err()$ is a function that estimates the error on a node being defined as

$$Err(t) = \frac{1}{n_t} \sum_{\langle \mathbf{x}_i, y_i \rangle \in D_t} (y_i - g(D_t))^2 \quad (2)$$

where D_t is the sample of cases in node t, n_t is the cardinality of this set, and $g(D_t)$ is a function of the cases in node t.

In standard regression trees, the function $g()$ is the average of the target variable Y, i.e., $\frac{1}{n_t} \sum_{\langle \mathbf{x}_i, y_i \rangle \in D_t} y_i$. This corresponds to assuming a constant model on each leaf of the tree. The evaluation of each candidate split requires obtaining the models at the respective left and right branches (Eq. 1). If this model is an average, rather efficient incremental algorithms can be used to evaluate all candidate splits. On the contrary, if $g()$ is a linear regression model or even other more complex models, this evaluation is not so simple, and it is computationally very demanding, as a result of which systems that use this strategy (Karalic 1992) become impractical for large problems. In this context, several authors have adopted the alternative of growing the trees assuming constant values in the leaves and then fitting the complex models on each of the obtained leaves (e.g. Quinlan 1992; Torgo 1997, 1999, 2000). This only requires fitting as many models as there are leaves in the final tree. The main drawback of this approach is that the splits for the tree nodes are selected assuming the leaves will have averages instead of the models that in effect will be used. This may lead to splits that are suboptimal for the models that will be fit on each leaf (Malerba et al. 2002; Malerba 2004). Several authors have tried to maintain the consistency of the split selection step with the models used in the leaves by proposing efficient algorithms for evaluating the different splits. In Malerba et al. (2002) and Malerba (2004) linear models are obtained in a stepwise manner during tree growth. In Chaudhuri et al. (1994), Loh (2002), and Dobra and Gehrke (2002) the computational complexity is reduced by transforming the original regression problem into a classification problem. In effect, the best split is chosen by looking at the distribution of the sign of the residuals of a linear model fitted locally. In Torgo (2002), Natarajan and Pednault (2002), and Vogel et al. (2007) the problem is addressed by proposing more efficient algorithms to evaluate all candidate splits. Finally, Potts and Sammut (2005) proposes an incremental algorithm to obtain model trees that fights the complexity of this task by imposing a limit on the number of splits that are considered for each node.

The most common form of model used in leaves is ▶ linear regression. Still, there are systems considering kernel models (Torgo 1997), local linear models (Torgo 1999), and partial linear models (Torgo 2000). These alternatives provide smoother function approximation, although with increased computational costs and less interpretable models.

▶ Pruning in model trees does not bring any additional challenges when compared to standard regression trees, and so similar methods are used for this over-fitting avoidance task. The same occurs with the use of model trees for obtaining predictions for new test cases. Each case is "dropped down" the tree from the root node, following the branches according to the logical tests in the nodes, till a leaf is reached. The model in this leaf is used to obtain the prediction for the test case.

Cross References

▶ Random Forests
▶ Regression
▶ Regression Trees
▶ Supervised Learning
▶ Training Data

Recommended Reading

Breiman L, Meisel WS (1976) General estimates of the intrinsic variability of data in nonlinear regression models. J Am Stat Assoc 71:301–307

Chaudhuri P, Huang M, Loh W, Yao R (1994) Piecewise-polynomial regression trees. Stat Sin 4:143–167

Dobra A, Gehrke JE (2002) Secret: a scalable linear regression tree algorithm. In: Proceedings of the 8th ACM SIGKDD international conference on knowledge discovery and data mining, Edmonton

Friedman J (1979) A tree-structured approach to nonparametric multiple regression. In: Gasser T, Rosenblatt M (eds) Smoothing techniques for curve estimation. Lecture notes in mathematics, vol 757. Springer, Berlin/New York, pp 5–22

Karalic A (1992) Employing linear regression in regression tree leaves. In Proceedings of ECAI-92, Vienna. Wiley & Sons

Loh W (2002) Regression trees with unbiased variable selection and interaction detection. Stat Sin 12:361–386

Malerba D, Appice A, Ceci M, Monopoli M (2002) Trading-off local versus global effects of regression nodes in model trees. In: ISMIS'02: proceedings of the 13th international symposium on foundations of intelligent systems, Lyon. Springer, pp 393–402

Malerba D, Esposito F, Ceci M, Appice A (2004) Top-down induction of model trees with regression and splitting nodes. IEEE Trans Pattern Anal Mach Intell 26(5):612–625

Natarajan R, Pednault E (2002) Segmented regression estimators for massive data sets. In: Proceedings of the second SIAM international conference on data mining (SDM'02), Arlington

Potts D, Sammut C (2005) Incremental learning of linear model trees. Mach Learn 61(1–3):5–48

Quinlan J (1992) Learning with continuous classes. In: Adams, Sterling (eds) Proceedings of AI'92, Hobart. World Scientific, pp 343–348

Torgo L (1997) Functional models for regression tree leaves. In: Fisher D (ed) Proceedings of the 14th international conference on machine learning, Nashville. Morgan Kaufmann Publishers

Torgo L (1999) Inductive learning of tree-based regression models. Ph.D. thesis, Faculty of Sciences, University of Porto

Torgo L (2000) Partial linear trees. In: Langley P (ed) Proceedings of the 17th International Conference on Machine Learning (ICML 2000), Stanford. Morgan Kaufmann Publishers, pp 1007–1014

Torgo L (2002) Computationally efficient linear regression trees. In: Jajuga K, Sokolowski A, Bock H (eds) Classification, clustering and data analysis: recent advances and applications (Proceedings of IFCS 2002). Studies in classification, data analysis, and knowledge organization. Springer, Berlin/New York, pp 409–415

Vogel D, Asparouhov O, Scheffer T (2007) Scalable look-ahead linear regression trees. In: KDD'07: proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, San Jose. ACM, pp 757–764

Model-Based Clustering

Arindam Banerjee and Hanhuai Shan
University of Minnesota, Minneapolis, MN, USA

Definition

Model-based clustering is a statistical approach to data clustering. The observed (multivariate) data is assumed to have been generated from a finite mixture of component models. Each component model is a probability distribution, typically a parametric multivariate distribution. For example, in a multivariate Gaussian mixture model, each component is a multivariate Gaussian distribution. The component responsible for generating a particular observation determines the cluster to which the observation belongs. However, the component generating each observation as well as the parameters for each of the component distributions are unknown. The key learning task is to determine the component responsible for generating each observation, which in turn gives the clustering of the data. Ideally, observations generated from the same component are inferred to belong to the same cluster. In addition to inferring the component assignment of observations, most popular learning approaches also estimate the parameters of each component in the process. The strength and popularity of the methods stem from the fact that they are applicable for a wide variety of data types, such as multivariate, categorical, sequential, etc., as long as suitable component generative models can be constructed. Such methods have found applications in several domains such as text clustering, image processing, computational biology, and climate sciences.

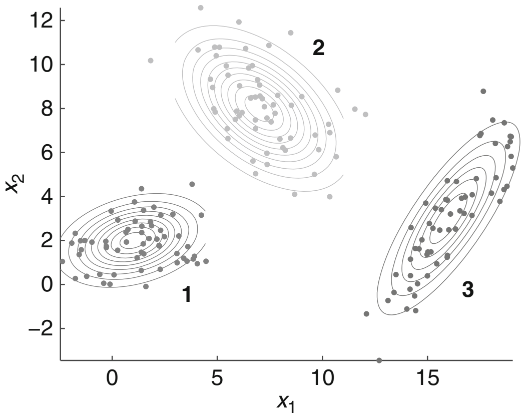
Structure of Learning System

Generative Model

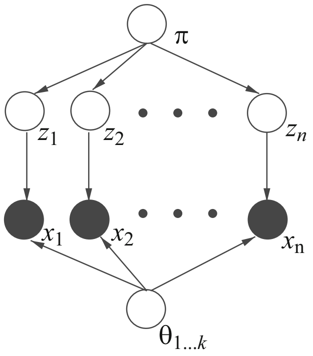
Let $X = \{x_1, \ldots, x_n\}$ be a dataset on which a k-clustering is to be performed. Let $p(x|\theta_1), \ldots, p(x|\theta_k)$ be k distributions which form the components of the mixture model from which the observed data is assumed to have been generated, and let $\pi = (\pi_1, \ldots, \pi_k)$ denote a prior distribution over the components. Then $\Theta = (\pi, \theta)$ constitutes the (unknown) parameters of the generative mixture model, where $\theta = \{\theta_1, \ldots, \theta_k\}$ and $\pi = \{\pi_1, \ldots, \pi_k\}$.

Given the model, an observation is assumed to be generated by the following two-step process: (1) randomly pick a component following the discrete distribution π over the components, i.e., the hth component is chosen with the probability of π_h; (2) the observation is sampled from the component distribution, e.g., if the hth component was chosen, we draw a sample $x \sim p(x|\ \theta_h)$. Each observation is assumed to be statistically independent so that they are all generated independently following the same two-step process.

Figure 1 gives an example of data drawn from a mixture of three ($k = 3$) 2-dimentional multivariate Gaussians. In the example, the discrete distribution over the component Gaussians is given by $\pi = (0.2, 0.3, 0.5)$. The parameter set $\theta_h, h = 1, 2, 3$ for any individual multivariate Gaussian consists of the mean vector μ_h and the



Model-Based Clustering, Fig. 1 Three 2-dimensional Gaussians



Model-Based Clustering, Fig. 2 Bayesian network for a finite mixture model

covariance matrix Σ_h. For the example, we have $\mu_1 = [1, 2], \mu_2 = [7, 8], \mu_3 = [16, 3]$, and $\Sigma_1 = \begin{bmatrix} .3 & 0.5196 \\ 0.5196 & 1 \end{bmatrix}$, $\Sigma_2 = \begin{bmatrix} 4 & -1.7321 \\ -1.7321 & 3 \end{bmatrix}$, $\Sigma_3 = \begin{bmatrix} 3 & 3.0984 \\ 3.0984 & 5 \end{bmatrix}$.

The generative process could be represented as a Bayesian network as shown in Fig. 2, where the arrows denote the dependencies among variables/parameters. In the Bayesian network, (π, θ) are the parameters of the mixture model, x_i are the observations and z_i are the latent variables corresponding to the component which generates x_i, $i = 1, \ldots, n$. To generate an observation x_i, the model first samples a latent variable z_i from the discrete distribution π, and then samples the observation x_i from component distribution $p(x|\theta_{zi})$.

Learning

Given a set of observations $X = \{x_1, \ldots, x_n\}$ assumed to have been generated from a finite mixture model, the learning task is to infer the latent variables z_i for each observation as well as estimate the model parameters $\Theta = (\pi, \theta)$. In the Gaussian mixture model example, the goal would be to infer the component responsible for generating each observation and estimate the mean and covariance for each component Gaussian as well as the discrete distribution π over the three Gaussians. After learning model parameters, the posterior probability $p(h|x_i, \Theta)$ of each observation x_i belonging to each component Gaussian gives a (soft) clustering for the observation.

The most popular approach for learning mixture models is based on maximum likelihood estimation (MLE) of the model parameters. In particular, given the set of observations X, one estimates the set of model parameters which maximizes the (log-)likelihood of observing the entire dataset X. For the finite mixture model, the likelihood of observing any data point x_i is given by

$$p(x_i|\Theta) = \sum_{h=1}^{k} \pi_h p(x_i|\theta_h). \qquad (1)$$

Since the data points in X are assumed to be statistically independent, the log-likelihood. (In practice, one typically focuses on maximizing the log-likelihood $\log p(X|\Theta)$ instead of the likelihood $p(X|\Theta)$ due to both numerical stability and analytical tractability). of observing the entire dataset X is given by

$$\log p(X|\Theta) = \log\left(\prod_{i=1}^{n} p(x_i|\pi, \theta)\right)$$
$$\sum_{i=1}^{n} \log\left(\sum_{h=1}^{k} \pi_h p(x_i|\theta_h)\right). \qquad (2)$$

A direct application of MLE is difficult since the log-likelihood cannot be directly optimized with respect to the model parameters. The standard approach to work around this issue is to use the expectation maximization (EM) algorithm which entails maximizing a tractable lower bound to the log-likelihood $\log p(X|\Theta)$. To this end, a latent variable z_i is explicitly introduced for each x_i to inform the component that x_i is generated from. The joint distribution of (x_i, z_i) is $p(x_i, z_i|\pi\theta) = \pi_{zi} p(x_i|\theta_{zi})$. Let $Z = \{z_1, \ldots, z_n\}$ denote the set of latent variables corresponding to $X = \{x_1, \ldots, x_n\}$. The joint log-likelihood of (X, Z) then becomes

$$\begin{aligned}\log p(X, Z|\Theta) &= \sum_{i=1}^{n} \log p(x_i, z_i|\Theta)\\ &= \sum_{i=1}^{n} (\log \pi_{zi} + \log p(x_i|\theta_{zi})).\end{aligned} \tag{3}$$

For a given set Z, it is easy to directly optimize (3) with respect to the parameters $\Theta = (\pi, \theta)$. However, Z is actually a random vector whose exact value is unknown. Hence, the log-likelihood $\log p(X, Z|\Theta)$ is a random variable depending on the distribution of Z. As a result, EM focuses on optimizing the following lower bound based on the expectation of $\log p(X, Z|\Theta)$ where the expectation is taken with respect to some distribution $p(Z)$ over the latent variable set Z. In particular, for any distribution $q(Z)$, we consider the lower bound

$$L(q, \Theta) = E_{z\sim q}[\log p(X, Z|\theta)] + H(q(Z)), \tag{4}$$

where the expectation on the first term is with respect to the posterior distribution $q(Z)$ and $H(q(Z))$ denotes the Shannon entropy of the latent variable set $Z \sim q(Z)$. A direct calculation shows that the difference between the true log-likelihood in (2) and the lower bound in (4) is exactly the relative entropy between $q(Z)$ and the posterior distribution $p(Z|X, \Theta)$, i.e.,

$$\begin{aligned}\log p(X|\Theta) - &L(q, \Theta)\\ &= KL(q(Z)\|p(Z|X, \Theta)) \geq 0\end{aligned} \tag{5}$$

$$\Rightarrow \log p(X|\Theta) \geq L(q, \Theta), \tag{6}$$

where $KL(||)$ denotes the KL-divergence or relative entropy. As a result, when $q(Z) = p(Z|X, \Theta)$, the lower bound is exactly equal to the log-likelihood $\log p(X|\Theta)$. EM algorithms for learning mixture models work by alternately optimizing the lower bound $L(q, \Theta)$ over q and Θ. Starting with an initial guess $\Theta^{(0)}$ of the parameters, in iteration t such algorithms perform the following two steps:

E-step Maximize $L(q, \Theta^{(t-1)})$ with respect to $q(Z)$ to obtain

$$\begin{aligned}q^{(t)}(Z) &= argmax_{q(Z)}\, L(q(Z), \Theta^{(t-1)})\\ &= p(Z|X, \Theta^{(t-1)}).\end{aligned} \tag{7}$$

M-step Maximize $L(q^{(t)}, \Theta)$ with respect to Θ, i.e.,

$$\Theta^{(t)} = arg_{\Theta} max L(q^{(t)}(Z), \Theta), \tag{8}$$

which is equivalent to

$$\Theta^{(t)} = argmax_{\Theta} \sum_{i=1}^{n} E_{zi}[\log p(x_i, z_i|\Theta)]$$

since the second term in (4) does not depend on Θ.

Model-based clustering of multivariate data is often performed by learning a Mixture of Gaussians (MoG) using the EM algorithm. In a MoG model, the parameters corresponding to each component are the mean and covariance for each Gaussian given by (μ_h, Σ_h), $h = 1, \ldots, k$. For a given dataset X, the EM algorithm for learning MoG starts with an initial guess $\Theta^{(0)}$ for the parameters where $\Theta^{(0)} = \{(\pi_h^{(0)}, \mu_h^{(0)}, \Sigma_h^{(0)}),$ $h = 1, \ldots, k\}$. At iteration t, the following updates are done:

E-step Update distributions over latent variables $z_i, i = 1, \ldots, n$ as

$$q^{(t)}(z_j = h) = p(z_j = h|x_j, \Theta^{(t-1)}) = \frac{\pi_h^{(t-1)} p(x_i|\mu_h^{(t-1)}, \Sigma_h^{(t-1)},)}{\Sigma_{h'=1}^{k} \pi_{h'}^{(t-1)} p(x_i|\mu_h^{(t-1)}, \Sigma_h^{(t-1)})}. \tag{9}$$

M-step Optimizing the lower bound over $\{(\pi_h, \mu_h, \Sigma_h), h = 1, \ldots, k\}$ yields

$$\pi_h^{(t)} = \frac{1}{n} \sum_{i=1}^{n} p(h|x_j,\ \Theta^{(t-1)}), \tag{10}$$

$$\mu_h^{(t)} = \frac{\Sigma_{i=1}^{n} x_i\, p(h|x_i, \Theta^{(t-1)})}{n\pi_h^{(t)}}, \tag{11}$$

$$\Sigma_h^{(t)} = \frac{\Sigma_{i=1}^{n} (x_i - \mu_h^{(t)})(x_i - \mu_h^{(t)})^T\, p(h|x_i, \Theta^{(t-1)})}{n\pi_h^{(t)}}. \tag{12}$$

The iterations are guaranteed to lead to monotonically non decreasing improvements of the lower bound $L(q, \Theta)$. The iterations are typically run till a suitable convergence criterion is satisfied. On convergence, one gets the estimates $\Theta = \{(\pi_h,\ \mu_h,\ \Sigma_h),\ h = 1,\ \ldots, k\}$ of the component parameters as well as the soft clustering $p(h|xi, \Theta)$ of individual data points. The alternating maximization algorithm outlined above can get stuck in a local minima or saddle point of the objective function. In general, the iterations are not guaranteed to converge to a global optima. In fact, different initializations $\Theta^{(0)}$ of parameters can yield different final results. In practice, one typically tries a set of different initializations and picks the best among them according to the final value of the lower bound obtained. Extensive empirical research has gone into devising good initialization schemes for EM algorithm in the context of learning mixture models.

Recent years have seen progress in the design and analysis of provably correct algorithms for learning mixture models for certain well behaved distributions, where the component distributions are assumed to be separated from each other in a well-defined sense. Such algorithms typically involve projecting data to a suitable lower-dimensional space where the components separate out and the clustering becomes simpler. One family of algorithms rely on random projections and are applicable to variety of problems including that of learning mixture of Gaussians. More recent developments include algorithms based on spectral projections and are applicable to any log-concave distri-butions.

Related Work

Model-based clustering is intimately related to a wide variety of centroid-based partitional clustering algorithms. In particular, the popular kmeans clustering algorithm can be viewed as a special case of learning mixture of Gaussians with a specific covariance structure. Given a dataset X, the kmeans problem is to find a partitioning $C = \{Ch, h = 1, \ldots, k\}$ of X such that the following objective is minimized:

$$J(C) = \sum_{h=1}^{k} \sum_{x \in C_h} \|x - \mu_h\|^2,$$

where μ_h is the mean of the points in C_h. Starting from an initial guess at the cluster means, the kmeans algorithm alternates between assigning points to the nearest cluster and updating the cluster means till convergence. Consider the problem of learning a mixture of Gaussians on X such that each Gaussian has a fixed covariance matrix $\sum_h = \beta I$, where I is the identity matrix and $\beta > 0$ is a constant. Then, as $\beta \to 0$, maximizing the scaled lower bound $\beta\ L(q, \Theta)$ corresponding to the mixture modeling problem becomes equivalent to minimizing the kmeans objective. Further, the EM algorithm outlined above reduces to the popular kmeans algorithm. In fact, such a reduction holds for a much larger class of centroid-based clustering algorithms based on Bregman divergences, which are a general class of divergence measures derived from convex function and have popular divergences such as squared Euclidean distance and KL-divergence as special cases. Centroid-based clustering with Bregman divergences can be viewed as a special case of learning mixtures of exponential family distributions with a reduction similar to the one from mixture of Gaussians to kmeans. Further, non linear clustering algorithms such as kernel kmeans can be viewed as a special case of learning mixture of Gaussians in a Hilbert space.

Recent years have seen generalizations of mixture models to mixed membership models and their non parametric extensions. Latent Dirichlet allocation is an example of such a mixed membership model for topic modeling in text corpora. The key novelty of mixed membership models is that they allow a different component proportions π_x for each observation x instead of a fixed proportion π as in mixture models. The added flexibility yields superior performance in certain problem domains.

Recommended Reading

Banerjee A, Merugu S, Dhillon I, Ghosh J (2005) Clustering with Bregman divergences. J Mach Learn Res 6:1705–1749

Bilmes J (1997) A gentle tutorial on the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. Technical Report ICSI-TR-97-02, University of Berkeley

Blei DM, Ng AY, Jordan MI (2003) Latent dirichlet allocation. J Mach Learn Res 3:993–1022

Dasgupta S (1999) Learning mixtures of Gaussians. In: IEEE symposium on foundations of Computer Science (FOCS). IEEE Press, Washington, DC

Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J R Stat Soc Ser B (Methodol) 39(1):1–38

Kannan R, Salmasian H, Vempala S (2005) The spectral method for general mixture models. In: Conference on learning theory (COLT)

McLachlan GJ, Krishnan T (1996) The EM algorithm and extensions. Wiley-Interscience, New York

McLachlan GJ, Peel D (2000) Finite mixture models. Wiley series in probability and mathematical statistics: applied probability and statistics section. Wiley, New York

Neal RM, Hinton GE (1998) A view of the EM algorithm that justifies incremental, sparse, and other variants. In: Jordan MI (ed) Learning in graphical models (pp 355–368). MIT Press, Cambridge, MA

Redner R, Walker H (1984) Mixture densities, maximum likelihood and the EM algorithm. SIAM Rev 26(2):195–239

Model-Based Control

▶ Internal Model Control

Model-Based Reinforcement Learning

Soumya Ray[1] and Prasad Tadepalli[2]
[1]Case Western Reserve University, Cleveland, OH, USA
[2]School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA

Synonyms

Indirect reinforcement learning

Definition

Model-based reinforcement learning refers to learning optimal behavior indirectly by learning a model of the environment by taking actions and observing the outcomes that include the next state and the immediate reward. The models predict the outcomes of actions and are used in lieu of or in addition to interaction with the environment to learn optimal policies.

Motivation and Background

▶ Reinforcement Learning (RL) refers to learning to behave optimally in a stochastic environment by taking actions and receiving rewards (Sutton and Barto 1998). The environment is assumed Markovian in that there is a fixed probability of the next state given the current state and the agent's action. The agent also receives an immediate reward based on the current state and the action. Models of the next-state distribution and the immediate rewards are referred to as "action models" and, in general, are not known to the learner. The agent's goal is to take actions, observe the outcomes including rewards and next states, and learn a policy or a mapping from states to actions that optimizes some performance measure. Typically the performance measure is the expected total reward in episodic domains and the expected average reward per time step or expected discounted total reward in infinite-horizon domains.

The theory of ▸ Markov Decision Processes (MDPs) implies that under fairly general conditions, there is a stationary policy, i.e., a time-invariant mapping from states to actions, which maximizes each of the above reward measures. Moreover, there are MDP solution algorithms, e.g., value iteration and policy iteration (Puterman 1994), which can be used to solve the MDP exactly given the action models. Assuming that the number of states is not exceedingly high, this suggests a straightforward approach for model-based reinforcement learning. The models can be learned by interacting with the environment by taking actions, observing the resulting states and rewards, and estimating the parameters of the action models through maximum likelihood methods. Once the models are estimated to a desired accuracy, the MDP solution algorithms can be run to learn the optimal policy.

One weakness of the above approach is that it seems to suggest that a fairly accurate model needs to be learned over the entire domain to learn a good policy. Intuitively it seems that we should be able to get by without learning highly accurate models for suboptimal actions. A related problem is that the method does not suggest how best to explore the domain, i.e., which states to visit and which actions to execute to quickly learn an optimal policy. A third issue is one of scaling these methods, including model learning, to very large state spaces with billions of states.

The remaining sections outline some of the approaches explored in the literature to solve these problems.

Theory and Methods

Systems that solve MDPs using value-based methods can take advantage of models in at least two ways. First, with an accurate model, they can use offline learning algorithms that directly solve the modeled MDPs. Second, in an online setting, they can use the estimated models to guide exploration and action selection. Algorithms have been developed that exploit MDP models in each of these ways. We describe some such algorithms below.

Common approaches to solving MDPs given a model are value or policy iteration (Sutton and Barto 1998; Kaelbling et al. 1996). In these approaches, the algorithms start with a randomly initialized value function or policy. In value iteration, the algorithm loops through the state space, updating the value estimates of each state using Bellman backups, until convergence. In policy iteration, the algorithm calculates the value of the current policy and then loops through the state space, updating the current policy to be greedy with respect to the backed up values. This is repeated until the policy converges.

When the model is unknown but being estimated as learning progresses, we could use value or policy iteration in the inner loop: after updating our current model estimate using an observed sample from the MDP, we could solve the updated MDP offline and take an action based on the solution. However, this is computationally very expensive. To gain efficiency, algorithms such as ▸ Adaptive Real-time Dynamic Programming (ARTDP) (Barto et al. 1995) and DYNA (Sutton 1990) perform one or more Bellman updates using the action models after each real-world action and corresponding update to either a state-based or state-action-based value function. Other approaches, such as prioritized sweeping (Moore and Atkeson 1993) and Queue-Dyna (Peng and Williams 1993), have considered the problem of intelligently choosing which states to update after each iteration.

A different approach to discovering the optimal policy is to use algorithms that calculate the gradient of the utility measure with respect to some adjustable policy parameters. The standard policy gradient approaches that estimate the gradient from immediate rewards suffer from high variance due to the stochasticity of the domain and the policy. Wang and Dietterich propose a model-based policy gradient algorithm that alleviates this problem by learning a partial model of the domain (Wang and Dietterich 2003). The partial model is solved to yield the value function of the current policy and the expected number of visits to each state, which are then used to derive the gradient of the policy in closed form. The authors observe that their approach converges in many fewer exploratory steps compared with model-free policy gradient algorithms in a number of domains including a real-world resource-controlled scheduling problem.

One of the many challenges in model-based reinforcement learning is that of efficient exploration of the MDP to learn the dynamics and the rewards. In the "Explicit Explore and Exploit" or E^3 algorithm, the agent explicitly decides between exploiting the known part of the MDP and optimally trying to reach the unknown part of the MDP (exploration) (Kearns and Singh 2002). During exploration, it uses the idea of "balanced wandering," where the least executed action in the current state is preferred until all actions are executed a certain number of times. In contrast, the R-Max algorithm implicitly chooses between exploration and exploitation by using the principle of "optimism under uncertainty" (Brafman and Tennenholtz 2002). The idea here is to initialize the model parameters optimistically so that all unexplored actions in all states are assumed to reach a fictitious state that yields maximum possible reward from then on regardless of which action is taken. Both these algorithms are guaranteed to find models whose approximate policies are close to the optimal with high probability in time polynomial in the size and mixing time of the MDP.

Since a table-based representation of the model is impractical in large state spaces, efficient model-based learning depends on compact parameterization of the models. Dynamic Bayesian networks offer an elegant way to represent action models compactly by exploiting conditional independence relationships and have been shown to lead to fast convergence of models (Tadepalli and Ok 1998). In some cases, choosing an appropriate prior distribution over model parameters can be important and lead to faster learning. In recent work, the acquisition of a model prior has been investigated in a multitask setting (Wilson et al. 2007). In this work, the authors use a hierarchical Bayesian model to represent classes of MDPs. Given observations from a new MDP, the algorithm uses the model to infer an appropriate class (creating a new class if none seem appropriate). It then uses the distributions governing the inferred class as a prior to guide exploration in the new MDP. This approach is able to significantly speed up the rate of convergence to optimal policy as more environments are seen.

In recent work, researchers have explored the possibility of using approximate models coupled with policy gradient approaches to solve hard control problems (Abbeel et al. 2006). In this work, the approximate model is used to calculate gradient directions for the policy parameters. When searching for an improved policy, however, the real environment is used to calculate the utility of each intermediate policy. Observations from the environment are also used to update the approximate model. The authors show that their approach improves upon model-based algorithms which only used the approximate model while learning.

Applications

In this section, we describe some domains where model-based reinforcement learning has been applied.

Model-based approaches have been commonly used in RL systems that play two-player games (Tesauro 1995; Baxter et al. 1998). In such systems, the model corresponds to legal moves in the game. Such models are easy to acquire and can be used to perform lookahead search on the game tree. For example, the TD-LEAF(λ) system (Baxter et al. 1998) uses the values at the leaves of an expanded game tree at some depth to update the estimate of the value of the current state. After playing a few hundred chess games, this algorithm was able to reach the play level of a US Master.

Model-based reinforcement learning has been used in a spoken dialog system (Singh et al.

1999). In this application, a dialog is modeled as a turn-based process, where at each step the system speaks a phrase and records certain observations about the response and possibly receives a reward. The system estimates a model from the observations and rewards and uses value iteration to compute optimal policies for the estimated MDP. The authors show empirically that, among other things, the system finds sensible policies and is able to model situations that involve "distress features" that indicate the dialog is in trouble.

It was shown that in complex real-world control tasks such as pendulum swing-up task on a real anthropomorphic robot arm, model-based learning is very effective in learning from demonstrations (Atkeson and Schaal 1997). A model is learned from the human demonstration of pendulum swing up, and an optimal policy is computed using a standard approach in control theory called linear quadratic regulation. Direct imitation of the human policy would not work in this case due to the small differences in the tasks and the imperfections of the robot controller. On the other hand, model-based learning was able to learn successfully from short demonstrations of pendulum swing up. However, on a more difficult swing-up task that includes pumping, model-based learning by itself was inadequate due to the inaccuracies in the model. They obtained better results by combining model-based learning with learning appropriate task parameters such as the desired pendulum target angle at an intermediate stage where the pendulum was at its highest point.

In more recent work, model-based RL has been used to learn to fly a remote-controlled helicopter (Abbeel et al. 2007). Again, the use of model-free approaches is very difficult, because almost any random exploratory action results in an undesirable outcome (i.e., a crash). To learn a model, the system bootstraps from a trajectory that is observed by watching an expert human fly the desired maneuvers. In each step, the system learns a model with the observed trajectory and finds a controller that works in simulation with the model. This controller is then tried with the real helicopter. If it fails to work well, the model is refined with the new observations and the process is repeated. Using this approach, the system is able to learn a controller that can repeatedly perform complex aerobatic maneuvers, such as flips and rolls.

Model-based RL has also been applied to other domains, such as robot juggling (Schaal and Atkeson 1994) and job-shop scheduling (Zhang and Dietterich 1995). Some work has also been done that compares model-free and model-based RL methods (Atkeson and Santamaria 1997). From their experiments, the authors conclude that, for systems with reasonably simple dynamics, model-based RL is more data efficient, finds better policies, and handles changing goals better than model-free methods. On the other hand, model-based methods are subject to errors due to inaccurate model representations.

Future Directions

Representing and learning richer action models for stochastic domains that involve relations, numeric quantities, and parallel, hierarchical, and durative actions is a challenging open problem. Efficient derivation of optimal policies from such rich representations of action models is another problem that is partially explored in ► symbolic dynamic programming. Constructing good policy languages appropriate for a given action model or class of models might be useful to accelerate learning near-optimal policies for MDPs.

Cross-References

- ► Adaptive Real-Time Dynamic Programming
- ► Autonomous Helicopter Flight Using Reinforcement Learning
- ► Bayesian Reinforcement Learning
- ► Efficient Exploration in Reinforcement Learning
- ► Symbolic Dynamic Programming

Recommended Reading

Abbeel P, Coates A, Quigley M, Ng AY (2007) An application of reinforcement learning to aerobatic helicopter flight. In: Advances in neural information processing systems, vol 19. MIT, Cambridge, pp 1–8

Abbeel P, Quigley M, Ng AY (2006) Using inaccurate models in reinforcement learning. In: Proceedings of the 23rd international conference on machine learning, Pittsburgh. ACM, New York, pp 1–8

Atkeson CG, Santamaria JC (1997) A comparison of direct and model-based reinforcement learning. In: Proceedings of the international conference on robotics and automation, Albuquerque. IEEE, pp 20–25

Atkeson CG, Schaal S (1997) Robot learning from demonstration. In: Proceedings of the fourteenth international conference on machine learning, Nashville, vol 4. Morgan Kaufmann, San Francisco, pp 12–20

Barto AG, Bradtke SJ, Singh SP (1995) Learning to act using real-time dynamic programming. Artif Intell 72(1):81–138

Baxter J, Tridgell A, Weaver L (1998) TDLeaf(λ): combining temporal difference learning with game-tree search. In: Proceedings of the ninth Australian conference on neural networks (ACNN'98), Brisbane, pp 168–172

Brafman RI, Tennenholtz M (2002) R-MAX – a general polynomial time algorithm for near-optimal reinforcement learning. J Mach Learn Res 2:213–231

Kaelbling LP, Littman ML, Moore AP (1996) Reinforcement learning: a survey. J Artif Intell Res 4:237–285

Kearns M, Singh S (2002) Near-optimal reinforcement learning in polynomial time. Mach Learn 49(2/3):209–232

Moore AW, Atkeson CG (1993) Prioritized sweeping: reinforcement learning with less data and less real time. Mach Learn 13:103–130

Peng J, Williams RJ (1993) Efficient learning and planning within the Dyna framework. Adapt Behav 1(4):437–454

Puterman ML (1994) Markov decision processes: discrete dynamic stochastic programming. Wiley, New York

Schaal S, Atkeson CG (1994) Robot juggling: implementation of memory-based learning. IEEE Control Syst Mag 14(1):57–71

Singh S, Kearns M, Litman D, Walker M (1999) Reinforcement learning for spoken dialogue systems. In: Advances in neural information processing systems, Denver, vol 11. MIT, pp 956–962

Sutton RS (1990) Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: Proceedings of the seventh international conference on machine learning, Austin. Morgan Kaufmann, San Francisco, pp 216–224

Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT, Cambridge

Tadepalli P, Ok D (1998) Model-based average-reward reinforcement learning. Artif Intell 100:177–224

Tesauro G (1995) Temporal difference learning and TD-Gammon. Commun ACM 38(3):58–68

Wang X, Dietterich TG (2003) Model-based policy gradient reinforcement learning. In: Proceedings of the 20th international conference on machine learning, Washington, DC. AAAI, pp 776–783

Wilson A, Fern A, Ray S, Tadepalli P (2007) Multi-task reinforcement learning: a hierarchical Bayesian approach. In: Proceedings of the 24th international conference on machine learning, Corvalis. Omnipress, Madison, pp 1015–1022

Zhang W, Dietterich TG (1995) A reinforcement learning approach to job-shop scheduling. In: Proceedings of the international joint conference on artificial intelligence, Montréal. Morgan Kaufman, pp 1114–1120

Modularity Detection

▶ Group Detection

MOO

▶ Multi-objective Optimization

Morphosyntactic Disambiguation

▶ POS Tagging

Most General Hypothesis

Synonyms

Maximally general hypothesis

Definition

A hypothesis, h, is a most general hypothesis if it covers none of the negative examples and there is no other hypothesis h' that covers no negative examples, such that h is strictly more specific than h'.

Cross-References

▶ Learning as Search

Most Similar Point

▶ Nearest Neighbor

Most Specific Hypothesis

Synonyms

Maximally specific hypothesis

Definition

A hypothesis, h, is a most specific hypothesis if it covers none of the negative examples and there is no other hypothesis h' that covers no negative examples, such that h is strictly more general than h'.

Cross-References

▶ Learning as Search

Multi-agent Learning

Yoav Shoham and Rob Powers
Stanford University, Stanford, CA, USA

Definition

Multi-agent learning (MAL) refers to settings in which multiple agents learn simultaneously. Usually defined in a game theoretic setting, specifically in repeated games or stochastic games, the key feature that distinguishes multi-agent learning from single-agent learning is that in the former the learning of one agent impacts the learning of others. As a result, neither the problem definition for multi-agent learning nor the algorithms offered follow in a straightforward way from the single-agent case. In this first of two entries on the subject, we focus on the problem definition.

Background

The topic of multi-agent learning (MAL henceforth) has a long history in game theory, almost as long as the history of game theory itself (another more recent term for the area within game theory is *interactive learning*). In artificial intelligence (AI), the history of *single*-agent learning is of course as rich if not richer; one need not look further than this encyclopedia for evidence. And while it is only in recent years that AI has branched into the multi-agent aspects of learning, it has done so with something of a vengeance. If in 2003 one could describe the AI literature on MAL by enumerating the relevant articles, today this is no longer possible. The leading conferences routinely feature articles on MAL, as do the journals (We acknowledge a simplification of history here. There is definitely MAL work in AI that predates the last few years, though the relative deluge is indeed recent. Similarly, we focus on AI since this is where most of the action is these days, but there are also other areas in computer science that feature MAL material; we mean to include that literature here as well).

While the AI literature maintains a certain flavor that distinguishes it from the game theoretic literature, the commonalities are greater than the differences. Indeed, alongside the area of mechanism design and perhaps the computational questions surrounding solution concepts such as the Nash equilibrium, MAL is today arguably one of the most fertile interaction grounds between computer science and game theory. The key aspect of MAL, which ties the work together and which distinguishes it from single-agent learning, is the fact that in MAL one cannot separate the process of learning from the process of teaching.

The learning of one agent causes it to change its behavior; this causes other agents to adapt their behavior, which in turn causes the first agent to keep adapting too. Such reciprocal – or interactive – learning calls not only for different types of learning algorithms but also for different yardsticks by which to evaluate learning. For this reason, the literature on MAL can be confusing. Not only do the learning techniques vary, but the goal of learning and the evaluation measures are diverse and often left only implicit.

We will couch our discussion in the formal setting of *stochastic games* (aka *Markov games*). Most of the MAL literature adopts this setting, and indeed most of it focuses on the even more narrow class of *repeated games*. Furthermore, stochastic games also generalize *Markov decision problems* (MDPs), the setting from which much of the relevant learning literature in AI originates. These are defined as follows.

A stochastic game can be represented as a tuple: $(N, S, \vec{A}, \vec{R}, T)$. N is a set of agents indexed $1, \ldots, n$. S is a set of n-agent stage games. $\vec{A} = A_1, \ldots, A_n$, with A_i the set of actions (or pure strategies) of agent i (note that we assume the agent has the same strategy space in all games; this is a notational convenience, but not a substantive restriction). $\vec{R} = R_1, \ldots, R_n$, with $R_i : S \times \vec{A} \to \mathcal{R}$ giving the immediate reward function of agent i for stage game S. $T : S \times \vec{A} \to \Pi(S)$ is a stochastic transition function, specifying the probability of the next stage game to be played based on the game just played and the actions taken in it.

We also need to define a way for each agent to aggregate the set of immediate rewards received in each state. For finitely repeated games, we can simply use the sum or average, while for infinite games, the most common approaches are to use either the limit average or the sum of discounted awards $\sum_{t=1}^{\infty} \delta^t r_t$, where r_t is the reward received at time t.

A repeated game is a stochastic game with only one stage game, while an MDP is a stochastic game with only one agent. (Note: While most of the MAL literature lives happily in this setting, we would be remiss not to acknowledge the literature that does not. Certainly, one could discuss learning in the context of extensive-form games of incomplete and/or imperfect information. Even farther afield, interesting studies of learning exist in large population games and evolutionary models, particularly *replicator dynamics (RD)* and *evolutionary stable strategies (ESS)*.)

What is there to learn in stochastic games? Here we need to be explicit about some aspects of stochastic games that were glossed over so far. Do the agents know the stochastic game, including the stage games and the transition probabilities? If not, do they at least know the specific game being played at each stage, or only the actions available to them? What do they see after each stage game has been played – only their own rewards, or also the actions played by the other agent(s)? Do they perhaps magically see the other agent(s)' mixed strategy in the stage game? And so on.

In general, games may be known or not, play may be observable or not, and so on. We will focus on known, fully observable games, where the other agent's strategy (or agents' strategies) is not known a priori (though in some cases, there is a prior distribution over it). In our restricted setting, there are two possible things to learn. First, the agent can learn the opponent's (or opponents') strategy (or strategies), so that the agent can then devise the best (or at least a good) response. Alternatively, the agent can learn a strategy of his own that does well against the opponents, without explicitly learning the opponent's strategy. The first is sometimes called *model-based learning* and the second *model-free learning*.

In broader settings, there is more to learn. In particular, with unknown games, one can learn the game itself. Some will argue that the restricted setting is not a true learning setting, but (a) much of the current work on MAL, particularly in game theory, takes place in this setting, and (b) the foundational issues we wish to tackle surface already here. In particular, our comments are intended to also apply to the work in the AI literature on games with unknown payoffs, work which builds on the success of learning in unknown MDPs. We will have more to say about the nature of "learning" in the setting of stochastic games in the following sections.

Problem Definition

When one examines the MAL literature, one can identify several distinct agendas at play, which are often left implicit and conflated. A prerequisite for success in the field is to be very explicit about the problem being addressed. Here we list five distinct coherent goals of MAL research. They each have a clear motivation and a success criterion. They can be caricatured as follows:

1. Computational
2. Descriptive
3. Normative
4. Prescriptive, cooperative
5. Prescriptive, noncooperative

The first agenda is computational in nature. It views learning algorithms as an iterative way to compute properties of the game, such as solution concepts. As an example, fictitious play was originally proposed as a way of computing a sample Nash equilibrium for zero-sum games, and replicator dynamics has been proposed for computing a sample Nash equilibrium in symmetric games. These tend not to be the most efficient computation methods, but they do sometimes constitute quick-and-dirty methods that can easily be understood and implemented.

The second agenda is descriptive – it asks how natural agents learn in the context of other learners. The goal here is to investigate formal models of learning that agree with people's behavior (typically, in laboratory experiments) or possibly with the behaviors of other agents (e.g., animals or organizations). This problem is clearly an important one and when taken seriously calls for strong justification of the learning dynamics being studied. One approach is to apply the experimental methodology of the social sciences.

The centrality of equilibria in game theory underlies the third agenda we identify in MAL, which for lack of a better term we called normative and which focuses on determining which sets of learning rules are in equilibrium with each other. More precisely, we ask which repeated game strategies are in equilibrium; it just so happens that in repeated games, most strategies embody a learning rule of some sort. For example, we can ask whether fictitious play and Q-learning, appropriately initialized, are in equilibrium with each other in a repeated Prisoner's Dilemma game.

The last two agendas are prescriptive; they ask how agents *should* learn. The first of these involves distributed control in dynamic systems. There is sometimes a need or desire to decentralize the control of a system operating in a dynamic environment, and in this case, the local controllers must adapt to each other's choices. This direction, which is most naturally modeled as a repeated or stochastic common-payoff (or "team") game. Proposed approaches can be evaluated based on the value achieved by the joint policy and the resources required, whether in terms of computation, communication, or time required to learn the policy. In this case, there is rarely a role for equilibrium analysis; the agents have no freedom to deviate from the prescribed algorithm.

In our final agenda, termed "prescriptive, noncooperative," we ask how an agent should act to obtain high reward in the repeated (and, more generally, stochastic) game. It thus retains the design stance of AI, asking how to design an optimal (or at least effective) agent for a given environment. It just so happens that this environment is characterized by the types of agents inhabiting it, agents who may do some learning of their own. The objective of this agenda is to identify effective strategies for environments of interest. An effective strategy is one that achieves a high reward in its environment, where one of the main characteristics of this environment is the selected class of possible opponents. This class of opponents should itself be motivated as being reasonable and containing opponents of interest. Convergence to an equilibrium is not a goal in and of itself.

Recommended Reading

Requisite background in game theory can be obtained from the many introductory texts and most compactly from Leyton-Brown and

Shoham (2008). Game theoretic work on multi-agent learning is covered in Fudenberg and Levine (1998) and Young (2004). An expanded discussion of the problems addressed under the header of MAL can be found in Shoham et al. (2007) and the responses to it in Vohra and Wellman (2007). Discussion of MAL algorithms, both traditional and more novel ones, can be found in the above references, as well as in Greenwald and Littman (2007).

Fudenberg D, Levine D (1998) The theory of learning in games. MIT, Cambridge

Greenwald A, Littman ML (eds) (2007) Special issue on learning and computational game theory. Mach Learn 67(1–2):3–6

Leyton-Brown K, Shoham Y (2008) Essentials of game theory. Morgan and Claypool, San Rafael

Shoham Y, Powers WR, Grenager T (2007) If multi-agent learning is the answer, what is the question? Artif Intell 171(1):365–377. Special issue on foundations of multiagent learning

Vohra R, Wellman MP (eds) (2007) Special issue on foundations of multiagent learning. Artif Intell 171(1)

Young HP (2004) Strategic learning and its limits. Oxford University Press, Oxford

Multi-agent Learning Algorithms

Yoav Shoham and Rob Powers
Stanford University, Stanford, CA, USA

Definition

Multi-agent learning (MAL) refers to settings in which multiple agents learn simultaneously. Usually defined in a game theoretic setting, specifically in repeated games or stochastic games, the key feature that distinguishes MAL from single-agent learning is that in the former the learning of one agent impacts the learning of others. As a result, neither the problem definition for multi-agent learning nor the algorithms offered follow in a straightforward way from the single-agent case. In this second of two entries on the subject, we focus on algorithms.

Some MAL Techniques

We will discuss three classes of techniques – one representative of work in game theory, one more typical of work in artificial intelligence (AI), and one that seems to have drawn equal attention from both communities.

Model-Based Approaches

The first approach to learning we discuss, which is common in the game theory literature, is the model-based one. It adopts the following general scheme:

1. Start with some model of the opponent's strategy.
2. Compute and play the best response.
3. Observe the opponent's play and update your model of his/her strategy.
4. Go to step 2.

Among the earliest, and probably the best-known, instance of this scheme is *fictitious play*. The model is simply a count of the plays by the opponent in the past. The opponent is assumed to be playing a stationary strategy, and the observed frequencies are taken to represent the opponent's mixed strategy. Thus after five repetitions of the Rochambeau game (R) in which the opponent played (R, S, P, R, P), the current model of his/her mixed strategy is $R = 0.4$, $P = 0.4$, $S = 0.2$.

There exist many variants of the general scheme, for example, those in which one does not play the exact best response in step 2. This is typically accomplished by assigning a probability of playing each pure strategy, assigning the best response the highest probability, but allowing some chance of playing any of the strategies. A number of proposals have been made of different ways to assign these probabilities such as *smooth fictitious play* and *exponential fictitious play*.

A more sophisticated version of the same scheme is seen in *rational learning*. The model is a distribution over the repeated-game strategies. One starts with some prior distribution; for example, in a repeated Rochambeau game, the prior could state that with probability 0.5 the opponent repeatedly plays the equilibrium strategy of the stage game, and, for all $k > 1$, with probability 2^{-k} she plays R k times and then reverts to the repeated equilibrium strategy. After each play, the model is updated to be the posterior obtained by Bayesian conditioning of the previous model. For instance, in our example, after the first non-R play of the opponent, the posterior places probability 1 on the repeated equilibrium play.

Model-Free Approaches

An entirely different approach that has been commonly pursued in the AI literature is the model-free one, which avoids building an explicit model of the opponent's strategy. Instead, over time one learns how well one's own various possible actions fare. This work takes place under the general heading of *reinforcement learning* (we note that the term is used somewhat differently in the game theory literature), and most approaches have their roots in the Bellman equations. We start our discussion with the familiar single-agent *Q-learning* algorithm for computing an optimal policy in an unknown Markov Decision Problem (MDP).

$$\begin{aligned} Q(s,a) &\leftarrow (1-\alpha_t)Q(s,a) + \alpha_t[R(s,a) \\ &\quad + \gamma V(s')] \\ V(s) &\leftarrow \max_{a \in A} Q(s,a). \end{aligned}$$

As is well known, with certain assumptions about the way in which actions are selected at each state over time and constraints on the learning rate schedule, α_t, Q-learning can be shown to converge to the optimal value function V^*.

The Q-learning algorithm can be extended to the multi-agent stochastic game setting by having each agent simply ignore the other agents and pretend that the environment is passive:

$$\begin{aligned} Q_i(s,a_i) &\leftarrow (1-\alpha_t)Q_i(s,a_i) + \alpha_t[R_i(s,\mathbf{a}) \\ &\quad + \gamma V_i(s')] \\ V_i(s) &\leftarrow \max_{a_i \in A_i} Q_i(s,a_i). \end{aligned}$$

Several authors have tested variations of the basic Q-learning algorithm for MAL. However, this approach ignores the multi-agent nature of the setting entirely. The Q-values are updated without regard for the actions selected by the other agents. While this can be justified when the opponents' distributions of actions are stationary, it can fail when an opponent may adapt its choice of actions based on the past history of the game.

The first step in addressing this problem is to define the Q-values as a function of all the agents' actions:

$$\begin{aligned} Q_i(s,\mathbf{a}) &\leftarrow (1-\alpha)Q_i(s,\mathbf{a}) \\ &\quad + \alpha[R_i(s,\mathbf{a}) + \gamma V_i(s')]. \end{aligned}$$

We are, however, left with the question of how to update V, given the more complex nature of the Q-values.

For (by definition, two-player) zero-sum stochastic games (SGs), the *minimax-Q* learning algorithm updates V with the minimax of the Q-values:

$$\begin{aligned} V_1(s) \leftarrow \max_{P_1 \in \Pi(A_1)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} P_1(a_1) \\ Q_1(s,(a_1,a_2)). \end{aligned}$$

Later work proposed other update rules for the Q and V functions focusing on the special case of common-payoff (or "team") games. A stage game is common-payoff if at each outcome all agents receive the same payoff. The payoff is, in general, different in different outcomes, and thus the agents' problem is that of coordination; indeed, these are also called *games of pure coordination.*

The work on zero-sum and common-payoff games continues to be refined and extended; much of this work has concentrated on probably optimal trade-offs between exploration and exploitation in unknown, zero-sum games. Another work attempted to extend the "Bellman heritage" to general-sum games (as opposed to zero-sum or common-payoff games), but the results here have been less conclusive.

Regret Minimization Approaches

Our third and final example of prior work in MAL is no-regret learning. It is an interesting example for two reasons. First, it has some unique properties that distinguish it from the work above. Second, both the AI and game theory communities appear to have converged on it independently. The basic idea goes back to early work on how to evaluate the success of learning rules in the mid-1950s and has since been extended and rediscovered numerous times over the years under the names of universal consistency, no-regret learning, and the Bayes' envelope. The following algorithm is a representative of this body of work. We start by defining the *regret*, $r_i^t(a_j, s_i)$ of agent i for playing the sequence of actions s_i instead of playing action a_j, given that the opponents played the sequence s_{-i}.

$$r_i^t(a_j, s_i | s_{-i}) = \sum_{k=1}^{t} R\left(a_j, s_{-i}^k\right) - R\left(s_i^k, s_{-i}^k\right).$$

The agent then selects each of its actions with probability proportional to max $\left(r_i^t(a_j, s_i), 0\right)$ at each time step $t + 1$.

Some Typical Results

One sees at least three kinds of results in the literature regarding the learning algorithms presented above and others similar to them. These are:

1. Convergence of the strategy profile to an (e.g., Nash) equilibrium of the stage game in self-play (i.e., when all agents adopt the learning procedure under consideration).
2. Successful learning of an opponent's strategy (or opponents' strategies).
3. Obtaining payoffs that exceed a specified threshold.

Each of these types comes in many flavors; here are some examples. The first type is perhaps the most common in the literature, in both game theory and AI. For example, while fictitious play does not in general converge to a Nash equilibrium of the stage game, the distribution of its play can be shown to converge to an equilibrium in zero-sum games, 2 × 2 games with generic payoffs, or games that can be solved by iterated elimination of strictly dominated strategies. Similarly in AI, minimax-Q learning is proven to converge in the limit to the correct Q-values for any zero-sum game, guaranteeing convergence to a Nash equilibrium in self-play. This result makes the standard assumptions of infinite exploration and the conditions on learning rates used in proofs of convergence for single-agent Q-learning.

Rational learning exemplifies results of the second type. The convergence shown is to correct beliefs about the opponent's repeated game strategy; thus it follows that, since each agent adopts a best response to their beliefs about the other agent, in the limit the agents will converge to a Nash equilibrium of the repeated game. This is an impressive result, but it is limited by two factors: the convergence depends on a very strong assumption of absolute continuity; and the beliefs converged to are correct only with respect to the aspects of history that are observable given the strategies of the agents. This is an involved topic, and the reader is referred to the literature for more details.

The literature on no-regret learning provides an example of the third type of result and has perhaps been the most explicit about the criteria for evaluating learning rules. For example, one pair of criteria that have been suggested is as follows. The first criterion is that the learning rule should be "safe," which is defined as the requirement that the learning rule must guarantee at least the minimax payoff of the game. (The minimax payoff is the maximum expected value a player can guarantee against any possible opponent.) The second criterion is that the rule should be "consistent." In order to be "consistent," the learning rule must guarantee that it does at least as well as the best response to the empirical distribution of play when playing against an opponent whose play is governed by independent draws from a fixed distribution. "Universal consistency" is then defined as the requirement that a learning rule does at least as well as the best response to the empirical distribution regardless of the actual strategy the opponent is employing (this implies both safety and consistency). The requirement of “universal consistency” is in fact equivalent to requiring that an algorithm exhibits *no-regret*, generally defined as follows, against all opponents.

$$\forall \epsilon > 0, \left(\lim_{t\to\inf}\left[\frac{1}{t}\max_{a_j\in A_i} r_i^t(a_j, s_i|s_{-i})\right] < \epsilon\right)$$

In both game theory and artificial intelligence, a large number of algorithms have been shown to satisfy universal consistency or no-regret requirements.

Recommended Reading

Requisite background in game theory can be obtained from the many introductory texts, and most compactly from Leyton-Brown and Shoham (2008). Game theoretic work on multiagent learning is covered in Fudenberg and Levine (1998) and Young (2004). An expanded discussion of the problems addressed under the header of MAL can be found in Shoham et al. (2007), and the responses to it in Vohra and Wellman (2007). Discussion of MAL algorithms, both traditional and more novel ones, can be found in the above references, as well as in Greenwald and Littman (2007).

Fudenberg D, Levine D (1998) The theory of learning in games. MIT, Cambridge

Greenwald A, Littman ML (eds) (2007) Special issue on learning and computational game theory. Mach Learn 67(1–2):3–6

Leyton-Brown K, Shoham Y (2008) Essentials of game theory. Morgan and Claypool, San Rafael

Shoham Y, Powers WR, Grenager T (2007) If multiagent learning is the answer, what is the question? Artif Intell 171(1):365–377. Special issue on foundations of multiagent learning

Vohra R, Wellman MP (eds) (2007) Special issue on foundations of multiagent learning. Artif Intell 171(1)

Young HP (2004) Strategic learning and its limits. Oxford University Press, Oxford

Multi-armed Bandit

▶ *k*-Armed Bandit

Multi-armed Bandit Problem

▶ *k*-Armed Bandit

MultiBoosting

Geoffrey I. Webb
Faculty of Information Technology, Monash
University, Victoria, Australia

Definition

MultiBoosting (Webb 2000) is an approach to ▶ multistrategy ensemble learning that combines features of ▶ AdaBoost and ▶ Bagging. The insight underlying MultiBoosting is that the primary effect of AdaBoost is ▶ bias reduction, while the primary effect of bagging is ▶ variance reduction. By combining the two techniques, it is possible to obtain both bias and variance reduction, the cumulative effect often being a greater reduction in error than can be obtained with the equivalent amount of computation by either AdaBoost or Bagging alone. Viewed from another perspective, as the size of an ensemble formed by either AdaBoost or Bagging is increased, each successive addition to the ensemble has decreasing effect. Thus, if the benefit of the first few applications of AdaBoost can be combined with the benefit of the first few applications of Bagging, the combined benefit may be greater than simply increasing the number of applications of one or the other.

Algorithm

MultiBoosting operates by dividing the ensemble of classifiers that is to be created into a number of subcommittees. Each of these subcommittees is formed by Wagging (Baner and Kohavi 1999), MultiBoosting. Tablel MultiBoost Algorithm

MultiBoost
input:

- S_0, a sequence of m labeled examples $\langle(x_1, y_1), \ldots, (x_m, y_m)\rangle$ with labels $y_j \in Y$.
- base learning algorithm *BaseLearn*.
- integer T specifying the number of iterations.
- vector of integers I_i specifying the iteration at which each subcommittee $i \geq 1$ should terminate.

```
1.  $S_1 = S_0$ with instance weights assigned to be 1.
2.  set $k = 1$.
3.  For $t = 1$ to $T$
4.    If $I_k = t$ then
5.    reweight $S_t$ .
6.    increment $k$.
7.  $C_t = BaseLearn(S')$.
8.  $\epsilon_t = \frac{\Sigma_{x_j \in S_t : C_t(x_j) \neq y_j} weight(x_j)}{m}$.
9.  if $\epsilon_t > 0.5$ then
10.   reweight $S_t$.
11.   increment $k$.
12.   go to 7.
13. otherwise if $\epsilon_t = 0$ then
14.   set $\beta_t$ to $10^{-10}$
15.   reweight $S_t$.
16.   increment $k$.
17. otherwise,
18.   $\beta_t = \frac{\epsilon_t}{(1-\epsilon_t)}$.
19.   $S_{t+1} = S_t$.
20.   For each $x_j \in S_{t+1}$,
21.     divide $weight\ (x_j)$ by $2\epsilon_t$ if $C_t(x_j) \neq y_j$ and
      $2\ (1 - \epsilon_t)$ otherwise.
22.     if $weight\ (x_j) < 10^{-8}$, set $weight\ (x_j)$ to $10^{-8}$
```

Output the final classifier: $C^*(x) = argmax_{y \in Y} \sum_{t; C_t(x)=y} log \frac{1}{\beta_t}$.

a variant of Bagging that utilizes weighted instances and, hence, is more readily integrated with AdaBoost. The ensemble is formed by applying AdaBoost to these subcommittees. The resulting algorithm is presented in Table 1. The learned ensemble classifier is C, and the tth member of the ensemble is C_t. Each S_t is a vector of n weighted training objects whose weights always sum to n. The weights change from turn to turn (the turns indicated by the subscript t). The base training algorithm *BaseLearn* should more heavily penalize errors on training instances with higher weights. ε_t is the weighted error of C_t on S_i. β_t is a weight assigned to the tth classifier, Ct. The operation reweight S_t sets the weights of the objects in S_t to random values drawn from the continuous Poisson distribution and then standardizes them to sum to n. The code set with a grey background is the code added to AdaBoost in order to create MultiBoost.

Cross-References

▶ AdaBoost
▶ Bagging
▶ Ensemble Learning
▶ Multistrategy Ensemble Learning

Recommended Reading

Bauer E, Kohavi R (1999) An empirical comparison of voting classification algorithms: bagging, boosting, and variants. Mach Learn 36(1):105–139

Webb GI (2000) MultiBoosting: a technique for combining boosting and wagging. Mach Learn 40(2):159–196

Multi-criteria Optimization

▶ Multi-objective Optimization

Multi-Instance Learning

Soumya Ray[1], Stephen Scott[2], and
Hendrik Blockeel[3,4]
[1]Case Western Reserve University, Cleveland, OH, USA
[2]University of Nebraska, Lincoln, NE, USA
[3]Katholieke Universiteit Leuven, Heverlee, Leuven, Belgium
[4]Leiden Institute of Advanced Computer Science, Heverlee, Belgium

Synonyms

Multiple-instance learning

Definition

Multiple-Instance (MI) learning is an extension of the standard supervised learning setting. In standard supervised learning, the input consists of a set of labeled instances each described by an attribute vector. The learner then induces a concept that relates the label of an instance to its attributes. In MI learning, the input consists of labeled examples (called "bags") consisting of *multisets* of instances, each described by an attribute vector, and there are constraints that relate the label of each bag to the unknown labels of each instance. The MI learner then induces a concept that relates the label of a bag to the attributes describing the instances in it. This setting contains supervised learning as a special case: if each bag contains exactly one instance, it reduces to a standard supervised learning problem.

Motivation and Background

The MI setting was introduced by Dietterich et al. (1997) in the context of drug activity prediction. Drugs are typically molecules that fulfill some desired function by binding to a target. If we wish to learn the characteristics responsible for binding, a possible representation of the problem is to represent each molecule as a set of low energy shapes or *conformations*, and describe each conformation using a set of attributes. Each such bag of conformations is given a label corresponding to whether the molecule is active or inactive. To learn a classification model, an algorithm assumes that every instance in a bag labeled negative is actually negative, whereas at least one instance in a bag labeled positive is actually positive with respect to the underlying concept.

From a theoretical viewpoint, MI learning occupies an intermediate position between standard propositional supervised learning and first-order relational learning. Supervised learning is a special case of MI learning, while MI learning is a special case of first-order learning. It has been argued that the MI setting is a key transition between standard supervised and relational learning DeRaedt (1998). At the same time, theoretical results exist that show that, under certain assumptions, certain concept classes that are probably approximately correct (PAC)-learnable (see PAC Learning) in a supervised setting remain PAC-learnable in an MI setting. Thus, the MI setting is able to leverage some of the rich representational power of relational learners while not sacrificing the efficiency of propositional learners. Figure 1 illustrates the relationships between standard supervised learning, MI learning, and relational learning.

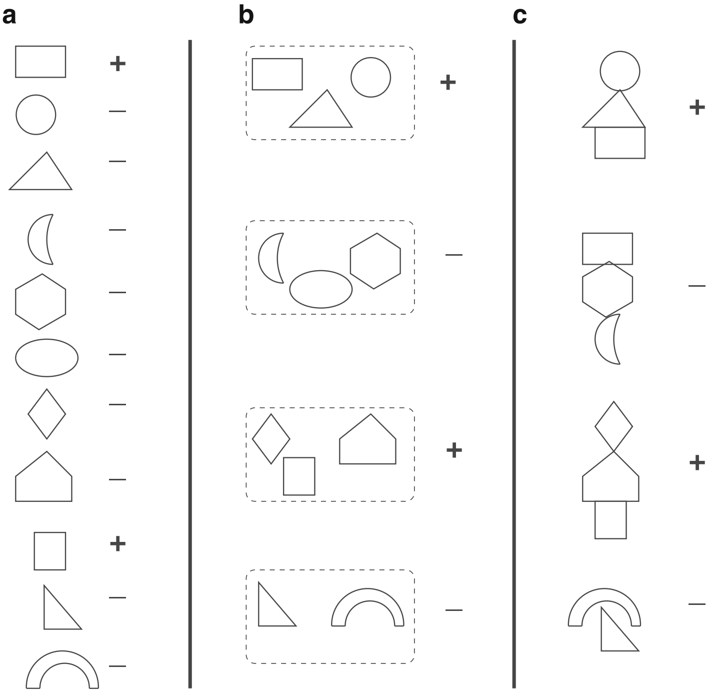
Since its introduction, a wide variety of tasks have been formulated as MI learning problems. Many new algorithms have been developed, and well-known supervised learning algorithms extended, to learn MI concepts. A great deal of work has also been done to understand what kinds of concepts can and cannot be learned efficiently in this setting. In the following sections, we discuss the theory, methods, and applications of MI learning in more detail.

Structure of the Problem

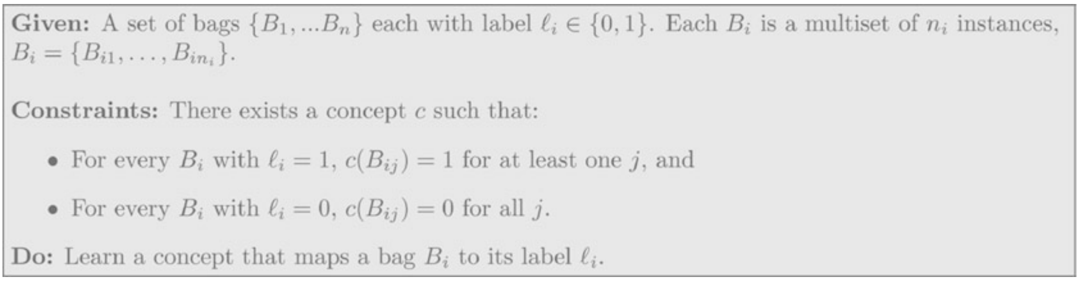
The general MI classification task in shown in Fig. 2. The MI regression task is defined analogously by substituting a real-valued response for the classification label. In this case, the constraint used by the learning algorithm is that the response of any bag is equal to the response of at least one of the instances in it, for example, it could be equal to the largest response over all the instances.

Notice the following problem characteristics:

- The number of instances in each bag can vary independently of other bags. This implies in particular that an MI algorithm must be able to handle bags with as few as one instance (this is a supervised learning setting) to bags with large numbers of instances.
- The number of instances in any positive bag that are "truly positive" could be many more than one – in fact, the definition does not rule



Multi-Instance Learning, Fig. 1 The relationship between supervised, multiple-instance (MI), and relational learning. (a) In supervised learning, each example (geometric figure) is labeled. A possible concept that explains the example labels shown is "the figure is a rectangle." (b) In MI learning, bags of examples are labeled. A possible concept that explains the bag labels shown is "the bag contains at least one figure that is a rectangle." (c) In relational learning, objects of arbitrary structure are labeled. A possible concept that explains the object labels shown is "the object is a stack of three figures and the bottom figure is a rectangle"



Given: A set of bags $\{B_1, ...B_n\}$ each with label $\ell_i \in \{0,1\}$. Each B_i is a multiset of n_i instances, $B_i = \{B_{i1}, \ldots, B_{in_i}\}$.

Constraints: There exists a concept c such that:

- For every B_i with $\ell_i = 1$, $c(B_{ij}) = 1$ for at least one j, and
- For every B_i with $\ell_i = 0$, $c(B_{ij}) = 0$ for all j.

Do: Learn a concept that maps a bag B_i to its label ℓ_i.

Multi-Instance Learning, Fig. 2 Statement of the multiple-instance classification problem

out the case where *all* instances in a positive bag are "truly positive."

- The problem definition does not specify how the instances in any bag are related to each other.

Theory and Methods

In this section we discuss some of the key algorithms and theoretical results in MI learning. We first discuss the methods and results for MI classification. Then we discuss the work on MI regression.

Multiple-Instance Classification

Axis-Parallel Rectangles (APRs) are a concept class that early work in MI classification focused on. These generative concepts specify upper and lower bounds for all numeric attributes describing each instance. An APR is said to "cover" an instance if the instance lies within it. An APR covers a bag if it covers at least one instance within it. The learning algorithm tries to find an APR such that it covers all positive bags and does not cover any negative bags.

An algorithm called "iterated-discrimination" was proposed by Dietterich et al. (1997) to learn APRs from MI data. This algorithm has two phases. In the first phase, it iteratively chooses a set of “relevant” attributes and grows an APR using this set. This phase results in the construction of a very “tight” APR that covers just positive bags. In the second phase, the algorithm expands this APR so that with high probability a new positive instance will fall within the APR. The key steps of the algorithm are outlined below. Note that initially, all attributes are considered to be “relevant.”

The algorithm starts by choosing a random instance in a positive bag. Let us call this instance I_1. The smallest APR covering this instance is a point. The algorithm then expands this APR by finding the smallest APR that covers any instance from a yet uncovered positive bag; call the newly covered instance I_2. This process is continued, identifying new instances $I_3, \ldots, I_k$, until all positive bags are covered. At each step, the APR is “backfitted” in a way that is reminiscent of the later Expectation-Maximization (EM) approaches: each earlier choice is revisited, and I_j is replaced with an instance from the same bag that minimizes the current APR (which may or may not be the same as the one that minimized it at step j).

This process yields an APR that imposes maximally tight bounds on all attributes and covers all positive bags. Based on this APR, a new set of “relevant” attributes is selected as follows. An attribute’s relevance is determined by how strongly it discriminates against negative instances, i.e., given the current APR bounds, how many negative instances the attribute excludes. Features are then chosen iteratively and greedily according to how relevant they are until all negative instances have been excluded. This yields a subset of (presumably relevant) attributes. The APR growth procedure in the previous paragraph is then repeated, with the size of an APR redefined as its size along relevant attributes only. The APR growth and attribute selection phases are repeated until the process converges.

The APR thus constructed may still be too tight, as it fits narrowly around the positive bags in the dataset. In the second phase of the algorithm, the APR bounds are further expanded using a kernel density estimate approach. Here, a probability distribution is constructed for each relevant attribute using Gaussian distributions centered at each instance in a positive bag. Then, the bounds on that attribute are adjusted so that with high probability, any positive instance will lie within the expanded APR.

Theoretical analyses of APR concepts have been performed along with the empirical approach, using Valiant’s “probably approximately correct” (PAC) learning model (Valiant 1984). In early work (Long and Tan 1998), it was shown that if each instance was drawn according to a fixed, unknown product distribution over the rational numbers, independently from every other instance, then an algorithm could PAC-learn APRs. Later, this result was improved in two ways (Auer et al. 1998). First, the restriction that the individual instances in each bag come from a product distribution was removed. Instead, each instance is generated by an arbitrary probability distribution (though each instance in a bag is still generated independently and identically distributed (iid) according to that one distribution). Second, the time and sample complexities for PAC-learning APRs were improved. Specifically, the algorithm described in this work PAC-learns APRs in

$$O\left(\frac{d^3n^2}{\epsilon^2}\log\frac{nd\log(1/\delta)}{\epsilon}\log\frac{d}{\delta}\right)$$

using

$$O\left(\frac{d^2n^2}{\epsilon^2}\log\frac{d}{\delta}\right)$$

time-labeled training bags. Here, d is the dimension of each instance, n is the (largest) number of instances per training bag, and ε and δ are parameters to the algorithm. A variant of this algorithm was empirically evaluated and found to be successful (Auer 1997).

Diverse Density (Maron 1998; Maron and Lozano-Pérez 1998) is a probabilistic generative framework for MI classification. The idea behind this framework is that, given a set of positive and negative bags, we wish to learn a concept that is “close” to at least one instance from each positive bag, while remaining “far” from every instance in every negative bag. Thus, the concept must describe a region of instance space that is "dense" in instances from positive bags, and is also "diverse" in that it describes every positive bag. More formally, let

$$DD(t) = \frac{1}{Z}\left(\prod_i \Pr(t|B_i^+)\prod_i \Pr(t|B_i^-)\right),$$

where t is a candidate concept, B_i^+ represents the ith positive bag, and B_i^- represents the ith negative bag. We seek a concept that maximizes $DD(t)$. The concept generates the instances of a bag, rather than the bag itself. To score a concept with respect to a bag, we combine t's probabilities for instances using a function based on noisy-OR Pearl (1998):

$$\Pr(t|B_i^+)\infty(1 - \prod_j (1 - \Pr(B_{ij}^+ \in t))) \quad (1)$$

$$\Pr(t|B_i^-)\infty \prod_j (1 - \Pr(B_{ij}^- \in t)) \quad (2)$$

Here, the instances B_{ij}^+ and B_{ij}^- belonging to t are the "causes" of the "event" that "t is the target." The concept class investigated by Maron (1998) is the class of generative Gaussian models, which are parameterized by the mean μ and a "scale" $s = \frac{1}{2\sigma^2}$:

$$\Pr(B_{ij} \in t)\infty e^{-\sum_k (s_k (B_{ijk} - \mu_k)^2)},$$

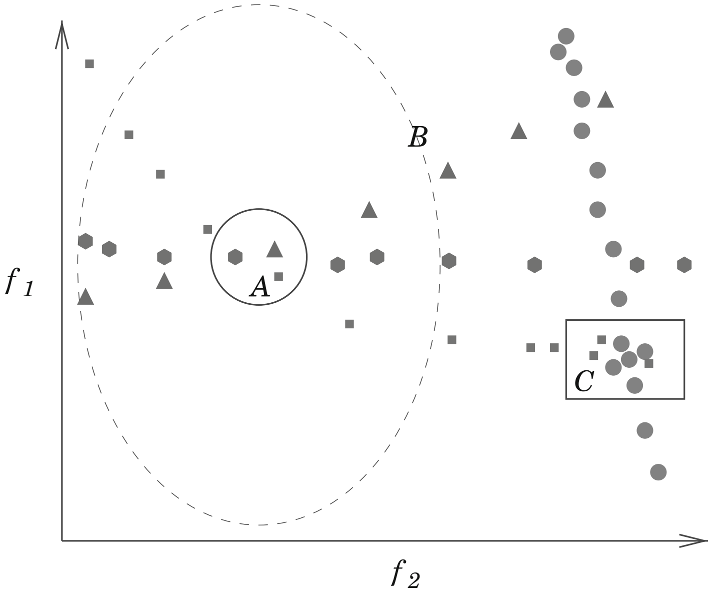
where k ranges over attributes. Figure 3 illustrates a concept that Diverse Density might learn when applied to an MI dataset.

Diverse Density with k disjuncts is a variant of Diverse Density that has also been investigated (Maron 1998). This is a class of disjunctive Gaussian concepts, where the probability of an instance belonging to a concept is given by the maximum probability of belonging to any of the disjuncts.

EM-DD (Zhang and Goldman 2001) is an example of a class of algorithms that try to identify the "cause" of a bag's label using EM. These algorithms sometimes assume that there is a single instance in each bag that is responsible for the bag's label (though variants using "soft EM" are possible). The key idea behind this approach is as follows: from each positive bag, we take a random instance and assume that this instance is the relevant one. We learn a hypothesis from these relevant instances and all negative bags. Next, for each positive bag, we replace the current relevant instance by the instance most consistent with the learned hypothesis (which will initially not be the chosen instance in general). We then relearn the hypothesis with these new instances. This process is continued until the set of chosen instances does not change (or alternatively, the objective function of the classifier reaches a fixed point). This procedure has the advantage of being computationally efficient, since the learning algorithm only uses one instance from each positive bag. This approach has also been used in MI regression described later.

"Upgraded" supervised learning algorithms can be used in a MI setting by suitably modifying their objective functions. Below, we summarize some of the algorithms that have been derived in this way.

1. ▶ Decision Tree induction algorithms have been adapted to the MI setting (Blockeel et al. 2005). The standard algorithm measures the quality of a split on an attribute by considering the class label distribution in the child nodes produced. In the MI case, this distribution is uncertain, because the true instance labels in positive bags are unknown. However, some rules have been identified that lead to empirically good MI trees: (1) use an asymmetric heuristic that favors early creation of pure positive (rather than negative) leaves, (2) once a positive leaf has been created, remove all other instances of the bags covered by this leaf; (3) abandon the depth-first or breadth-first order in which nodes are usually split, adopting a best-first strategy instead (indeed, because of (2), the result of tree learning is now sensitive to the order in which the nodes are split).
2. ▶ Artificial Neural Networks have been adapted to the MI setting by representing



Multi-Instance Learning, Fig. 3 An illustration of the concept that Diverse Density searches for on a simple MI dataset with three positive bags and one negative bag, where each instance (represented by the geometric figures) is described by two attributes, f_1 and f_2. Each type of figure represents one bag, i.e., all triangles belong to one bag, all circles belong to a second bag, and so forth. The bag containing the red circles is negative, while the other bags are positive. Region C is a region of high density, because several instances belong to that region. Region A is a region of high "Diverse Density," because several instances *from different positive bags* belong to that region, and no instances from negative bags are nearby. Region B shows a concept that might be learned if the learning algorithm assumed that all instances in every positive bag are positive (Figure adapted from Maron 1998)

the bag classifier as a network that combines several copies of a smaller network, which represents the instance classifier, with a smooth approximation of the *max* combining function (Ramon and DeRaedt 2000). Weight update rules for a backpropagation algorithm working on this network have been derived. Later work on MI neural networks has been performed independently by others (Zhou and Zhang 2002).

3. ▶ Logistic Regression has been adapted to the MI setting by using it as an instance-based classifier and combining the instance-level probabilities using functions like softmax (Ray and Craven 2005) and arithmetic and geometric averages (Xu and Frank 2004).
4. The k-Nearest Neighbor algorithm has been adapted to the MI setting by using set-based distance metrics, such as variants based on the Hausdorff distance. However, this alone does not solve the problem – it is possible for a positive bag to be mistakenly classified negative if it contains a "true negative" instance that happens to be much closer to negative instances in other negative bags. To solve this, a "Citation-kNN" (Wang and Zucker 2000) approach has been proposed that also considers, for each bag B, the labels of those bags for which B is a nearest neighbor.
5. ▶ Support Vector Machines have been adapted to the MI setting in several ways. In one method, the constraints in the quadratic program for SVMs is modified to account for the fact that certain instance labels are unknown but have constraints relating them (Andrews et al. 2003). In another method, new kernels are designed for MI data by modifying standard supervised SVM kernels (Gartner et al. 2002) or designing new kernels (Tao et al. 2004). The modification allows these MI ker-

nels to distinguish between positive and negative bags if the supervised kernel could distinguish between ("true") positive and negative instances.

6. ► Rule learning algorithms have been adapted to the MI setting in two ways. One method has investigated upgrading a supervised rule-learner, the ripper system (Cohen 1995), to the MI setting by modifying its objective function to account for bags and addressing several issues that resulted. Another method has investigated using general purpose relational algorithms, such as foil (Quinlan 1990) and tilde (Blockeel and De Raedt 1998), and providing them with an appropriate ► inductive bias so that they learn the MI concepts. Further, it has been observed that techniques from MI learning can also be used inside relational learning algorithms (Alphonse and Matwin 2002).

A large-scale empirical analysis of several such propositional supervised learning algorithms and their MI counterparts has been performed (Ray and Craven 2005). This analysis concludes that (1) no single MI algorithm works well across all problems. Thus, different inductive biases are suited to different problems, (2) some MI algorithms consistently perform better than their supervised counterparts but others do not (hence for these biases there seems room for improvement), and (3) assigning a larger weight to false positives than to false negatives is a simple but effective method to adapt supervised learning algorithms to the MI setting. It was also observed that the advantages of MI learners may be more pronounced if they would be evaluated on the task of labeling individual instances rather than bags.

Along with "upgrading" supervised learning algorithms, a *theoretical analysis of supervised learners* learning with MI data has been carried out (Blum and Kalai 1998). In particular, the MI problem has been related to the problem of learning in the presence of classification noise (i.e., each training example's label is flipped with some probability $< 1/2$). This implies that any concept class that is PAC-learnable in the presence of such noise is also learnable in the MI learning model when each instance of a bag is drawn iid. Since many concept classes are learnable under this noise assumption (using e.g., *statistical queries* Kearns 1998), Blum and Kalai's result implies PAC learnability of many concept classes. Further, they improved on previous learnability results (Auer et al. 1998) by reducing the number of training bags required for PAC learning by about a factor of n with only an increase in time complexity of about $\log n/\varepsilon$.

Besides these positive results, a *negative learnability result* describing when it is hard to learn concepts from MI data is also known (Auer et al. 1998). Specifically, if the instances of each bag are allowed collectively to be generated according to an arbitrary distribution, learning from MI examples is as hard as PAC-learning disjunctive normal form (DNF) formulas from single-instance examples, which is an open problem in learning theory that is believed to be hard. Further, it has been showed that if an efficient algorithm exists for the non-iid case that outputs as its hypothesis an axis-parallel rectangle, then NP = RP (Randomized Polynomial time, see e.g., Papadimitriou 1994), which is very unlikely.

Learning from structured MI data has received some attention (McGovern and Jensen 2003). In this work, each instance is a graph, and a bag is a set of graphs (e.g., a bag could consist of certain subgraphs of a larger graph). To learn the concepts in this structured space, the authors use a modified form of the Diverse Density algorithm discussed above. As before, the concept being searched for is a point (which corresponds to a graph in this case). The main modification is the use of the size of the maximal common subgraph to estimate the probability of a concept – i.e., the probability of a concept given a bag is estimated as proportional to the size of the maximal common subgraph between the concept and any instance in the bag.

Multiple-Instance Regression

Regression problems in an MI setting have received less attention than the classification problem. Two key directions have been explored in this setting. One direction extends the well-known standard ▸ linear regression method to the MI setting. The other direction considers extending various MI classification methods to a regression setting.

In *MI Linear Regression* (Ray and Page 2001) (referred to as multiple-instance regression in the cited work), it is assumed that the hypothesis underlying the data is a linear model with Gaussian noise on the value of the dependent variable (which is the response). Further, it is assumed that it is sufficient to model one instance from each bag, i.e., that there is some *primary* instance which is responsible for the real-valued label. Ideally, one would like to find a hyperplane that minimizes the squared error with respect to these primary instances. However, these instances are unknown during training. The authors conjecture that, given enough data, a good approximation to the ideal is given by the "best-fit" hyperplane, defined as the hyperplane that minimizes the training set squared error by fitting one instance from each bag such that the response of the fitted instance most closely matches the bag response. This conjecture will be true if the nonprimary instances are not a better fit to a hyperplane than the primary instances. However, exactly finding the "best-fit" hyperplane is intractable. It is shown that the decision problem "Is there a hyperplane which perfectly fits one instance from each bag?" is *NP*-complete for arbitrary numbers of bags, attributes, and at most three instances per bag. Thus, the authors propose an approximation algorithm which iterates between choosing instances and learning linear regression models that best fit them, similar to the EM-DD algorithm described earlier.

Another direction has explored *extending MI classification algorithms* to the regression setting. This approach (Dooly et al. 2002) uses algorithms like Citation-kNN and Diverse Density to learn real-valued concepts. To predict a real value, the approach uses the average of the nearest neighbor responses or interprets the Gaussian "probability" as a real number for Diverse Density.

Recent work has analyzed the Diverse Density-based regression in the *online* model (Angluin 1988; Littlestone 1988) (see ▸ online learning). In the online model, learning proceeds in *trials*, where in each trial a single example is selected adversarially and given to the learner for classification. After the learner predicts a label, the true label is revealed and the learner incurs a *loss* based on whether its prediction was correct. The goal of the online learner is to minimize the loss over all trials. Online learning is harder than PAC learning in that there are some PAC-learnable concept classes that are not online learnable.

In the regression setting above (Dooly et al. 2006), there is a point concept, and the label of each bag is a function of the distance between the concept and the point in the bag closest to the target. It is shown that similar to Auer et al.'s lower bound, learning in this setting using labeled bags alone is as hard as learning DNF. They then define an *MI membership query* (MI-MQ) in which an adversary defines a bag $B = \{p_1, \ldots, p_n\}$ and the learner is allowed to ask an oracle for the label of bag $B + \vec{v} = \{p_1 + \vec{v}, \ldots, p_n + \vec{v}\}$ for any d-dimensional vector $\vec{v}$. Their algorithm then uses this MI-MQ oracle to online learn a real-valued MI concept in time $O(dn^2)$.

Applications

In this section, we describe domains where MI learning problems have been formulated.

Drug activity was the motivating application for the MI representation (Dietterich et al. 1997). Drugs are typically molecules that fulfill some desired function by binding to a target. In this domain, we wish to predict how strongly a given molecule will bind to a target. Each molecule is a three-dimensional entity and takes on multiple shapes or *conformations* in solution. We know that for every molecule showing activity, at least one of its low energy conformations possesses the right shape for interacting with the target. Similarly, if the molecule does not show drug-like activity, none of its conformations possess the right shape for interaction. Thus, each molecule is represented as a bag, where each instance is a low energy conformation of the molecule. A well-known example from this domain is the MUSK dataset. The positive class in this data consists of molecules that smell "musky." This dataset has two variants, MUSK1 and MUSK2, both with similar numbers of bags, with MUSK2 having many more instances per bag.

Content-Based Image Retrieval is another domain where the MI representation has been used (Maron and Lozano-Pérez 1998; Zhang et al. 2002). In this domain, the task is to find images that contain objects of interest, such as tigers, in a database of images. An image is represented by a bag. An instance in a bag corresponds to a segment in the image, obtained by some segmentation technique. The underlying assumption is that the object of interest is contained in (at least) one segment of the image. For example, if we are trying to find images of mountains in a database, it is reasonable to expect most images of mountains to have certain distinctive segments characteristic of mountains. An MI learning algorithm should be able to use the segmented images to learn a concept that represents the shape of a mountain and use the learned concept to collect images of mountains from the database.

The *identification of protein families* has been framed as an MI problem (Tao et al. 2004). The objective in that work is to classify given protein sequences according to whether they belong to the family of thioredoxin-fold proteins. The given proteins are first aligned with respect to a motif that is known to be conserved in the members of the family. Each aligned protein is represented by a bag. A bag is labeled positive if the protein belongs to the family, and negative otherwise. An instance in a bag corresponds to a position in a fixed length sequence around the conserved motif. Each position is described by a vector of attributes; each attribute describes the properties of the amino acid at that position, and is smoothed using the same properties from its neighbors.

Text Categorization is another domain that has used the MI representation (Andrews et al. 2003; Ray and Craven 2005). In this domain, the task is to classify a document as belonging to a certain category or not. Often, whether the document belongs to the specified category is the function of a few passages in the document. These passages are however not labeled with the category information. Thus, a document could be represented as a set of passages. We assume that each positive document (i.e., that belongs to the specified category) has at least one passage that contains words that indicate category membership. On the other hand, a negative document (that does not belong to the category) has no passage that contain words indicating category membership. This formulation has been used to classify whether MEDLINE documents should be annotated with specific MeSH terms (Andrews et al.) and to determine if specific documents should be annotated with terms from the Gene Ontology (Ray and Craven 2005).

Time-series data from the hard drives have been used to define an MI problem (Murray et al. 2005). The task here is to distinguish drives that fail from others. Each hard drive is a bag. Each instance in the bag is a fixed-size window over timepoints when the drive's state was measured using certain attributes. In the training set, each drive is labeled according to whether it failed during a window of observation. An interesting aspect to prediction in this setting is that it is done online, i.e., the algorithm learns a classifier for instances, which is applied to each instance as it becomes available in time. The authors learn a naïve Bayes model using an EM-based approach to solve this problem.

Discovering useful subgoals in reinforcement learning has been formulated as an MI problem (McGovern and Barto 2001). Imagine that a robot has to get from one room to another by passing through a connecting door. If the robot knew of the existence of the door, it could decompose the problem into two simpler subproblems to be solved separately: getting from the initial location in the first room to the door, and then getting from the door to its destination. How could the robot discover such a "useful subgoal?" One approach formulates this as an MI problem. Each trajectory of the robot, where the robot starts at the source and then moves for some number of time steps, is considered to be a bag. An instance in a bag is a state of the world, that records observations such as, "is the robot's current location a door?" Trajectories that reach the destination are positive, while those that do not are negative. Given this data, we can learn a classifier that predicts which states are more likely to be seen on successful trajectories than on unsuccessful ones. These states are taken to be useful subgoals. In the previous example, the MI algorithm could learn that the state "location is a door" is a useful subgoal, since it appears on all successful trajectories, but infrequently on unsuccessful ones.

Future Directions

MI learning remains an active research area. One direction that is being explored relaxes the "Constraints" in Fig. 2 in different ways (Tao et al. 2004; Weidmann et al. 2003). For example, one could consider constraints where at least a certain number (or fraction) of instances have to be positive for a bag to be labeled positive. Similarly, it may be the case that a bag is labeled positive only if it does not contain a specific instance. Such relaxations are often studied as "generalized multiple-instance learning."

One such generalization of MI learning has been formally studied under the name "geometric patterns." In this setting, the target concept consists of a collection of APRs, and a bag is labeled positive if and only if (1) each of its points lies in a target APR, and (2) every target APR contains a point. Noise-tolerant PAC algorithms (Goldman and Scott 1999) and online algorithms (Goldman et al. 2001) have been presented for such concept classes. These algorithms make no assumptions on the distribution used to generate the bags (e.g., instances might not be generated by an iid process). This does not violate Auer et al.'s lower bound since these algorithms do not scale with the dimension of the input space.

Another recent direction explores the connections between MI and semi-supervised learnings. Semi-supervised learning generally refers to learning from a setting where some instance labels are unknown. MI learning can be viewed as one example of this setting. Exploiting this connection between MI learning and other methods for semi-supervised learning, recent work (Rahmani and Goldman 2006) proposes an approach where an MI problem is transformed into a semi-supervised learning problem. An advantage of the approach is that it automatically also takes into account unlabeled bags.

Cross-References

▶ Artificial Neural Networks
▶ Attribute
▶ Classification
▶ Data Set
▶ Decision Tree
▶ Expectation Maximization Clustering
▶ First-Order Logic
▶ Gaussian Distribution
▶ Inductive Logic Programming
▶ Kernel Methods
▶ Linear Regression
▶ Nearest Neighbor
▶ Noise
▶ Online Learning
▶ PAC Learning
▶ Relational Learning
▶ Supervised Learning

Recommended Reading

Alphonse E, Matwin S (2002) Feature subset selection and inductive logic programming. In: Proceedings of the 19th international conference on machine learning, pp 11–18. Morgan Kaufmann, San Francisco

Andrews S, Tsochantaridis I, Hofmann T (2003) Support vector machines for multiple-instance learning. In Becker S, Thrun S, Obermayer K (eds) Advances in neural information processing systems, vol 15. MIT Press, Cambridge, MA, pp 561–568

Angluin D (1988) Queries and concept learning. Mach Learn 2(4):319–342

Auer P (1997) On learning from multi-instance examples: empirical evaluation of a theoretical approach. In: Proceeding of 14th international conference on machine learning, pp 21–29. Morgan Kaufmann, San Francisco

Auer P, Long PM, Srinivasan A (1998) Approximating hyper-rectangles: learning and pseudorandom sets. J Comput Syst Sci 57(3):376–388

Blockeel H, De Raedt L (1998) Top-down induction of first order logical decision trees. Artif Intell 101(1–2):285–297

Blockeel H, Page D, Srinivasan A (2005) Multi-instance tree learning. In: Proceedings of 22nd international conference on machine learning, Bonn, pp 57–64

Blum A, Kalai A (1998) A note on learning from multiple-instance examples. Mach Learn J 30(1):23–29

Cohen WW (1995) Fast effective rule induction. In: Proceedings of the 12th international conference on machine learning. Morgan Kaufmann, San Francisco

DeRaedt L (1998) Attribute-value learning versus inductive logic programming: the missing links. In: Proceedings of the eighth international conference on inductive logic programming. Springer, New York, pp 1–8

Dietterich T, Lathrop R, Lozano-Perez T (1997) Solving the multiple-instance problem with axis-parallel rectangles. Artif Intell 89(1–2):31–71

Dooly DR, Goldman SA, Kwek SS (2006) Real-valued multiple-instance learning with queries. J Comput Syst Sci 72(1):1–15

Dooly DR, Zhang Q, Goldman SA, Amar RA (2002) Multiple-instance learning of real-valued data. J Mach Learn Res 3:651–678

Gartner T, Flach PA, Kowalczyk A, Smola AJ (2002) Multi-instance kernels. In: Sammut C, Hoffmann A (eds) Proceedings of the 19th international conference on machine learning, pp 179–186. Morgan Kaufmann, San Francisco

Goldman SA, Kwek SK, Scott SD (2001) Agnostic learning of geometric patterns. J Comput Syst Sci 6(1):123–151

Goldman SA, Scott SD (1999) A theoretical and empirical study of a noise-tolerant algorithm to learn geometric patterns. Mach Learn 37(1):5–49

Kearns M (1998) Efficient noise-tolerant learning from statistical queries. J ACM 45(6):983–1006

Long PM, Tan L (1998) PAC learning axis-aligned rectangles with respect to product distributions from multiple-instance examples. Mach Learn 30(1):7–21

Littlestone N (1988) Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. Mach Learn 2(4):285–318

Maron O (1998) Learning from ambiguity. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA

Maron O, Lozano-Pérez T (1998) A framework for multiple-instance learning. In: Jordan MI, Kearns MJ, Solla SA (eds) Advances in neural information processing systems, vol 10. MIT Press, Cambridge, MA, pp 570–576

McGovern A, Barto AG (2001) Automatic discovery of subgoals in reinforcement learning using diverse density. In: Proceedings of the 18th international conference on machine learning. Morgan Kaufmann, San Francisco, pp 361–368

McGovern A, Jensen D (2003) Identifying predictive structures in relational data using multiple instance learning. In: Proceedings of the 20th international conference on machine learning. AAAI Press, Menlo Park, pp 528–535

Murray JF, Hughes GF, Kreutz-Delgado K (2005) Machine learning methods for predicting failures in hard drives: A multiple-instance application. J Mach Learn Res 6:783–816

Papadimitriou C (1994) Computational complexity. Addison-Wesley, Boston

Pearl J (1998) Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann, San Mateo

Quinlan JR (1990) Learning logical definitions from relations. Mach Learn 5:239–266

Rahmani R, Goldman SA (2006) MISSL: Multiple-instance semi-supervised learning. In: Proceedings of the 23rd international conference on machine learning, pp 705–712. ACM Press, New York

Ramon J, DeRaedt L (2000) Multi instance neural networks. In: Proceedings of ICML-2000 workshop on attribute-value and relational learning

Ray S, Craven M (2005) Supervised versus multiple-instance learning: an empirical comparison. In: Proceedings of the 22nd international conference on machine learning. ACM Press, New York, pp 697–704

Ray S, Page D (2001) Multiple instance regression. In: Proceedings of the 18th international conference on machine learning. Morgan Kaufmann, Williamstown

Tao Q, Scott SD, Vinodchandran NV (2004) SVM-based generalized multiple-instance learning via approximate box counting. In: Proceedings of the 21st international conference on machine learning. Morgan Kaufmann, San Francisco, pp 779–806

Valiant LG (1984) A theory of the learnable. Commun ACM 27(11):1134–1142

Wang J, Zucker JD (2000) Solving the multiple-instance problem: a lazy learning approach. In: Proceedings of the 17th international conference on machine learning. Morgan Kaufmann, San Francisco, pp 1119–1125

Weidmann N, Frank E, Pfahringer B (2003) A two-level learning method for generalized multi-instance problems. In: Proceedings of the European conference on machine learning. Springer, Berlin/Heidelberg, pp 468–479

Xu X, Frank E (2004) Logistic regression and boosting for labeled bags of instances. In: Proceedings of the Pacific-Asia conference on knowledge discovery and data mining, Sydney, pp 272–281

Zhang Q, Goldman S (2001) EM-DD: an improved multiple-instance learning technique. In: Advances in Neural Information Processing Systems. MIT Press, Cambridge, MA, pp 1073–1080

Zhang Q, Yu W, Goldman S, Fritts J (2002) Content-based image retrieval using multiple-instance learning. In: Proceedings of the 19th international conference on machine learning. Morgan Kaufmann, San Francisco, pp 682–689

Zhou ZH, Zhang ML (2002) Neural networks for multi-instance learning. Technical Report, Nanjing University, Nanjing

Multi-label Learning

Zhi-Hua Zhou[1] and Min-Ling Zhang[2]
[1]National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
[2]School of Computer Science and Engineering, Southeast University, Nanjing, China

Abstract

Multi-label learning is an important machine learning setting where each example is associated with multiple class labels simultaneously. Firstly, definition, motivation and background, and learning system structure for multi-label learning are introduced. Secondly, multi-label evaluation measures and the issue of label correlation are discussed. Thirdly, basic ideas and technical details on four representative multi-label learning algorithms are considered. Lastly, theory, extensions, and future challenges on multi-label learning are introduced.

Definition

Multi-label learning is an extension of the standard supervised learning setting. In contrast to standard supervised learning where one training example is associated with a single class label, in multi-label learning, one training example is associated with *multiple* class labels simultaneously. The multi-label learner induces a function that is able to assign multiple proper labels (from a given label set) to unseen instances. Multi-label learning reduces to standard supervised learning by restricting the number of class labels per instance to one.

Motivation and Background

Most classification learning approaches treat the class values as disjoint – each object may belong only to a single class, such as *on* or *off*. Some applications, however, have categories that are not mutually exclusive – a single object may belong to multiple classes (Zhang and Zhou 2014). For instance, in text categorization, a news document on presidential election can cover multiple topics such as *politics*, *economics*, *diplomacy*, and *TV debate* (Schapire and Singer 2000); in image classification, a natural scene image can contain multiple sceneries such as the *sky*, *sea*, *boat*, and *beach* (Boutell et al. 2004). Actually, multi-label objects are often encountered in many applications such as bioinformatics, multimedia content annotation, information retrieval, and web mining (Zhang and Zhou 2014).

The goal of multi-label learning is to induce a function that can predict a subset of labels for an unseen instance from a given label set. Research into this important problem emerged in early 2000 and significant research progress has followed (Zhang and Zhou 2014).

Structure of Learning System

Let $\mathcal{X} = \mathcal{R}^d$ denote the d-dimensional instance space and $\mathcal{Y} = \{y_1, y_2, \ldots, y_q\}$ denote the label space consisting of q class labels. Given the multi-label training set $\mathcal{D} = \{(\boldsymbol{x}_i, Y_i) \mid 1 \le i \le m\}$, the task of multi-label learning is to learn a function $h : \mathcal{X} \mapsto 2^{\mathcal{Y}}$ mapping from the instance space to the *powerset* of the label space. For each multi-label training example $(\boldsymbol{x}_i, Y_i)$, $\boldsymbol{x}_i \in \mathcal{X}$ is a d-dimensional feature vector and $Y_i \subseteq \mathcal{Y}$ is the set of class labels associated with $\boldsymbol{x}_i$. The learned function $h(\cdot)$ predicts the proper label set for any unseen instance $\boldsymbol{x}$ as $h(\boldsymbol{x}) \subseteq \mathcal{Y}$.

An alternative model to $h(\cdot)$ returned by most multi-label learning systems corresponds to a real-valued function $f : \mathcal{X} \times \mathcal{Y} \mapsto \mathbb{R}$. Here, $f(\boldsymbol{x}, y)$ can be regarded as the predictive *confidence* of $y \in \mathcal{Y}$ being a proper label for $\boldsymbol{x}$. In other words, for the multi-label example $(\boldsymbol{x}, Y)$, the predictive output $f(\boldsymbol{x}, y')$ on *relevant* label

$y' \in Y$ should be larger than the predictive output $f(\boldsymbol{x}, y'')$ on *irrelevant* label $y'' \notin Y$, i.e., $f(\boldsymbol{x}, y') > f(\boldsymbol{x}, y'')$. By referring to a threshold function $t : \mathcal{X} \mapsto \mathbb{R}$, $h(\cdot)$ can be derived from the real-valued function $f(\cdot, \cdot)$ by: $h(\boldsymbol{x}) = \{y \mid f(\boldsymbol{x}, y) > t(\boldsymbol{x}), y \in \mathcal{Y}\}$.

Evaluation Measures

In standard supervised learning, popular measures used to evaluate the learning performance include *accuracy*, *precision*, *recall*, etc. In multi-label learning, however, these single-label evaluation measures cannot be adopted directly due to the multi-label nature of the data. Therefore, a number of evaluation measures have been designed for multi-label learning. These measures can be roughly categorized into two groups, i.e., *example-based* measures and *label-based* measures (Zhang and Zhou 2014). Example-based measures evaluate the generalization performance of the learned multi-label predictor on each test example separately and then return the mean value across the test set; label-based measures evaluate the generalization performance of the predictor on each class label separately and then return the macro-/micro-averaging value across all class labels.

Let $\mathcal{S} = \{(\boldsymbol{x}_i, Y_i) \mid 1 \leq i \leq p\}$ denote the multi-label test set, and $h(\cdot)$ (or equivalently $f(\cdot, \cdot)$) denote the learned multi-label predictor. Typical example-based measures include:

- *Subset Accuracy*: $\frac{1}{p}\sum_{i=1}^{p}[\![h(\boldsymbol{x}_i) = Y_i]\!]$. This measure evaluates the proportion of test examples whose predicted label set coincides with the ground-truth label set. Here, $[\![\pi]\!]$ returns 1 if predicate π holds and 0 otherwise.
- *Hamming Loss*: $\frac{1}{p}\sum_{i=1}^{p}\frac{1}{q}|h(\boldsymbol{x}_i)\Delta Y_i|$. This measure evaluates the proportion of misclassified instance-label pairs, i.e., a relevant label is missed or an irrelevant label is predicted. Here, Δ stands for the symmetric difference between two sets and $|\cdot|$ measures the cardinality of a set.
- *One-Error*: $\frac{1}{p}\sum_{i=1}^{p}[\![\arg\max_{y\in\mathcal{Y}} f(\boldsymbol{x}_i, y) \notin Y_i]\!]$. This measure evaluates the proportion of test examples whose top-1 predicted label fails to be a relevant label.
- *Coverage*: $\frac{1}{p}\sum_{i=1}^{p}\max_{y\in Y_i} rank_f(\boldsymbol{x}_i, y) - 1$. This measure evaluates the number of steps needed to move down the ranked label list so as to cover all relevant labels of the test example. Here, $rank_f(\boldsymbol{x}, y)$ returns the rank of class label y within label space $\mathcal{Y}$ according to the descending order specified by $f(\boldsymbol{x}, \cdot)$.
- *Ranking Loss*: $\frac{1}{p}\sum_{i=1}^{p}\frac{1}{|Y_i||\bar{Y}_i|}|\{(y', y'')| f(\boldsymbol{x}, y') \leq f(\boldsymbol{x}_i, y''),\ (y', y'') \in Y_i \times \bar{Y}_i\}|$. This measure evaluates the proportion of incorrectly ordered label pairs, i.e., an irrelevant label yields larger output value than a relevant label. Here, $\bar{Y}_i$ is the complementary set of Y_i in $\mathcal{Y}$.
- *Average Precision*: $\frac{1}{p}\sum_{i=1}^{p}\frac{1}{|Y_i|}\sum_{y\in Y_i} \frac{|\{y' | rank_f(\boldsymbol{x}_i, y') \leq rank_f(\boldsymbol{x}_i, y), y' \in Y_i\}|}{rank_f(\boldsymbol{x}_i, y)}$. This measure evaluates the average proportion of labels ranked higher than a relevant label $y \in Y_i$ that are also relevant.

For *hamming loss*, *one-error*, *coverage*, and *ranking loss*, the smaller the value, the better the generalization performance. For other example-based measures, the larger the value, the better the performance.

For label-based measures, to characterize the binary classification performance of the predictor on each label $y_j \in \mathcal{Y}$, four basic quantities regarding the test examples are commonly used: TP_j (#true positive), FP_j (#false positive), TN_j (#true negative), and FN_j (#false negative). It is evident that most binary classification measures can be derived based on these quantities. Let $B(TP_j, FP_j, TN_j, FN_j)$ denote a certain binary classification measure, label-based multi-label measures can be defined in either of the following ways:

- *Macro-B*: $\frac{1}{q}\sum_{j=1}^{q} B(TP_j, FP_j, TN_j, FN_j)$. This multi-label measure is derived by assuming equal importance for each label.
- *Micro-B*: $B(\sum_{j=1}^{q} TP_j, \sum_{j=1}^{q} FP_j, \sum_{j=1}^{q} TN_j, \sum_{j=1}^{q} FN_j)$. This multi-label measure is derived by assuming equal importance for each example.

Among popular choices of $B \in \{accuracy, precision, recall, F\}$, the larger the macro-/micro-B value, the better the performance.

Label Correlation

The major challenge of learning from multi-label data lies in the potentially tremendous-sized output space. Here, the number of possible *label sets* to be predicted grows exponentially as the number of class labels increases. For example, a label space with a moderate number of 20 class labels will lead to more than 1 million (i.e., 2^{20}) possible label sets. Thus, many label sets will rarely have examples appearing in the training set, leading to poor performance if they are learned separately.

To deal with the challenge of huge output space, a common practice is to exploit the *label correlation* to facilitate the learning process (Zhang and Zhou 2014). For instance, the probability of an image having label *Africa* would be high if we know it already has labels *grassland* and *lions*; a document is unlikely to be labeled as *recreation* if it is related to *legislation* and *police*. Actually, the fundamental issue distinguishing multi-label learning from traditional supervised learning lies in the fact that in multi-label learning it is crucial to exploit the label relations.

A widely-used strategy is to estimate the correlation among labels directly from the training examples based on the assumed correlation model. Based on the *order of correlations* being modeled, the estimation techniques can be roughly categorized into three categories:

(a) First-order techniques tackling multi-label learning task in a *label-by-label* style and thus ignoring the coexistence of other labels, such as decomposing the multi-label learning problem into a number of independent binary classification problems (one per label) (Boutell et al. 2004; Zhang and Zhou 2007)
(b) Second-order techniques tackling multi-label learning task by considering *pairwise* correlations between labels, such as the ranking between relevant and irrelevant labels (Elisseeff and Weston 2002; Schapire and Singer 2000)
(c) High-order techniques tackling multi-label learning task by considering high-order correlations among labels, such as assuming the correlations among all labels (Read et al. 2011) or random subsets of labels (Tsoumakas et al. 2011)

Another strategy is to adopt domain knowledge about label relations as input to the multi-label learning algorithms. One conventional source of domain knowledge corresponds to the label hierarchies (or taxonomies) available in some applications such as text classification (Rousu et al. 2005). There is also a recent strategy which tries to discover and exploit label relations during the procedure of learning the multi-label predictors (Zhang and Zhou 2014).

Learning Algorithms

To design learning algorithms for multi-label data, two complementary philosophies naturally arise. On one hand, *algorithm adaptation* methods work by fitting algorithms to data, i.e., adapting popular standard supervised learning algorithms to deal with multi-label data. On the other hand, *problem transformation* methods work by fitting data to algorithms, i.e., transforming multi-label data to accommodate other well-established learning frameworks. During the past decade, lots of algorithms have been developed following these philosophies (Zhang and Zhou 2014). This section briefly introduces four representative algorithms, including algorithm adaptation methods ML-KNN (multi-label k-nearest neighbor) (Zhang and Zhou 2007) and RANK-SVM (ranking support vector machine) (Elisseeff and Weston 2002), as well as problem transformation methods CC (classifier chain) (Read et al. 2011) and RAKEL (random k-labelsets) (Tsoumakas et al. 2011). These algorithms are simply chosen to manifest the essentials of two key design philosophies, which by no means exclude the importance of other multi-label learning algorithms.

ML-KNN adapts the *k-nearest neighbor* technique to deal with multi-label data (Zhang and Zhou 2007). Specifically, the *maximum a posteriori* (MAP) rule is utilized to make prediction for unseen instance by reasoning with the labeling information from its neighbors. Given the multi-label training set $\mathcal{D}$ and unseen instance $\boldsymbol{x}$, let $\mathcal{N}(\boldsymbol{x})$ denote the set of k nearest neighbors of $\boldsymbol{x}$ identified in $\mathcal{D}$. Accordingly, the following statistics can be calculated based on the labeling information of the neighbors in $\mathcal{N}(\boldsymbol{x})$: $C_j = \sum_{(\boldsymbol{x}_i, Y_i) \in \mathcal{N}(\boldsymbol{x})} [\![y_j \in Y_i]\!]$. Namely, C_j records the number of neighbors which take the j-th class label y_j as their relevant label. Let $P(H_j \mid C_j)$ represent the posterior probability that the event of H_j (i.e., $\boldsymbol{x}$ has y_j as its relevant label) holds under the condition of C_j (i.e., $\boldsymbol{x}$ has exactly C_j neighbors with relevant label y_j). Similarly, let $P(\neg H_j \mid C_j)$ represent the posterior probability that H_j does not hold under the same condition. Based on the MAP rule, the label set for $\boldsymbol{x}$ is predicted by

$$Y = \{y_j \mid P(H_j \mid C_j) > P(\neg H_j \mid C_j),\ 1 \le j \le q\} \qquad (1)$$

According to the Bayes rule, we have $P(H_j \mid C_j) \propto P(H_j) \cdot P(C_j \mid H_j)$ and $P(\neg H_j \mid C_j) \propto P(\neg H_j) \cdot P(C_j \mid \neg H_j)$. Therefore, it suffices to make prediction by estimating the prior probabilities $\{P(H_j),\ P(\neg H_j)\}$ and the likelihoods $\{P(C_j \mid H_j),\ P(C_j \mid \neg H_j)\}$. These probabilistic terms can be estimated from the training set via the *frequency counting* strategy (Zhang and Zhou 2007). In general, ML-KNN assumes label independence in its learning procedure and optimizes the evaluation measure of hamming loss (or equivalently macro-/micro-accuracy).

RANK-SVM adapts *large margin* methods to deal with multi-label data (Elisseeff and Weston 2002). Specifically, a set of linear classifiers are optimized to minimize the empirical ranking loss. Given the learning system with q linear classifiers $\mathcal{W} = \{(\boldsymbol{w}_j, b_j) \mid 1 \le j \le q\}$, its margin over each multi-label training example $(\boldsymbol{x}_i, Y_i)$ corresponds to

$$\gamma_i = \min_{(y_j, y_k) \in Y_i \times \bar{Y}_i} \frac{\langle \boldsymbol{w}_j - \boldsymbol{w}_k, \boldsymbol{x}_i \rangle + b_j - b_k}{||\boldsymbol{w}_j - \boldsymbol{w}_k||} \qquad (2)$$

Here, $\langle \cdot, \cdot \rangle$ returns the inner product between two vectors. Conceptually, Eq. (2) considers the signed L_2-distance of $\boldsymbol{x}_i$ to the decision hyperplane of every relevant-irrelevant label pair (y_j, y_k): $\langle \boldsymbol{w}_j - \boldsymbol{w}_k, \boldsymbol{x} \rangle + b_j - b_k = 0$, and then returns the minimum as the margin on $(\boldsymbol{x}_i, Y_i)$. Accordingly, the margin of the learning system on the whole training set $\mathcal{D}$ is $\min_{(\boldsymbol{x}_i, Y_i) \in \mathcal{D}} \gamma_i$. Under the ideal case that the learning system can properly rank every relevant-irrelevant label pair for each training example, the large margin optimization problem turns out to be

$$\begin{aligned} \min_{\mathcal{W}} \quad & \max_{1 \le j < k \le q} ||\boldsymbol{w}_j - \boldsymbol{w}_k||^2 \\ \text{s.t.}: \quad & \langle \boldsymbol{w}_j - \boldsymbol{w}_k, \boldsymbol{x}_i \rangle + b_j - b_k \ge 1 \\ & 1 \le i \le m,\ (y_j, y_k) \in Y_i \times \bar{Y}_i \end{aligned} \qquad (3)$$

By approximating `max` by `sum` and introducing slack variables to accommodate violated constraints, Eq. (3) can be re-formulated as

$$\begin{aligned} \min_{\{\mathcal{W}, \Xi\}} \quad & \sum_{j=1}^{q} ||\boldsymbol{w}_j||^2 + C \sum_{i=1}^{m} \frac{1}{|Y_i||\bar{Y}_i|} \sum_{(y_j, y_k) \in Y_i \times \bar{Y}_i} \xi_{ijk} \\ \text{s.t.}: \quad & \langle \boldsymbol{w}_j - \boldsymbol{w}_k, \boldsymbol{x}_i \rangle + b_j - b_k \ge 1 - \xi_{ijk} \\ & \xi_{ijk} \ge 0,\ 1 \le i \le m,\ (y_j, y_k) \in Y_i \times \bar{Y}_i \end{aligned} \qquad (4)$$

Here, $\Xi = \{\xi_{ijk} \mid 1 \le i \le m, (y_j, y_k) \in Y_i \times \bar{Y}_i\}$ is the set of slack variables. The first objective term in Eq. (4) corresponds to the margin of the learning system, whereas the second objective term corresponds to the empirical ranking loss. The solution to Eq. (4) can be found by invoking standard *quadratic programming* (QP) procedure in its primal form or incorporating kernel trick in its dual form. The label set for unseen instance is predicted by thresholding the output of each classifier in $\mathcal{W}$. In general, RANK-SVM assumes second-order label correlations (relevant-irrelevant label pair) in its learning procedure and optimizes the evaluation measure of ranking loss.

CC transforms the multi-label learning problem into a chain of *binary* classification problems. Specifically, subsequent classifiers in the chain are built upon the predictions of preceding ones. Without loss of generality, suppose all the class labels in $\mathcal{Y}$ are ordered in a chain: $y_1 \succ y_2 \succ \cdots \succ y_q$. For the j-th class label y_j in the chain, a corresponding binary training set can be constructed by taking the relevancy of each preceding label as an extra feature to the instance:

$$\mathcal{D}_j = \left\{\left([\boldsymbol{x}_i, \mathbf{pre}^i_j], \phi(Y_i, y_j)\right) \mid 1 \leq i \leq m\right\}$$
$$\text{where } \mathbf{pre}^i_j = (\phi(Y_i, y_1), \ldots, \phi(Y_i, y_{j-1}))^{\mathrm{T}} \tag{5}$$

Here, $\phi(Y, y) = [\![y \in Y]\!]$ represents the binary assignment of class label y w.r.t. label set Y. As shown in Eq. (5), each instance $\boldsymbol{x}_i$ is appended with an extra feature vector $\mathbf{pre}^i_j$ representing the relevancy of those labels preceding y_j. After that, a binary classifier $g_j : \mathcal{X} \times \{0,1\}^{j-1} \mapsto \{0,1\}$ can be induced for y_j by utilizing some binary learning algorithm $\mathcal{B}$, i.e., $g_j \hookleftarrow \mathcal{B}(\mathcal{D}_j)$. For unseen instance $\boldsymbol{x}$, its label set is predicted by traversing the classifier chain iteratively. The predicted binary assignment of y_j on $\boldsymbol{x}$, denoted as $\lambda^{\boldsymbol{x}}_j$, are recursively determined by

$$\lambda^{\boldsymbol{x}}_1 = g_1(\boldsymbol{x})$$
$$\lambda^{\boldsymbol{x}}_j = g_j([\boldsymbol{x}, \lambda^{\boldsymbol{x}}_1, \ldots, \lambda^{\boldsymbol{x}}_{j-1}]) \quad (2 \leq j \leq q) \tag{6}$$

Therefore, the predicted label set corresponds to: $Y = \{y_j \mid \lambda^{\boldsymbol{x}}_j = 1,\ 1 \leq j \leq q\}$. Evidently, the chaining order over the class labels has significant influence on the effectiveness of CC. To account for the effect of chaining order, an *ensemble* of classifier chains can be built with diverse random chaining orders. In general, CC assumes high-order label correlations (among all labels) in its learning procedure and optimizes the evaluation measure of hamming loss (or equivalently macro-/micro-accuracy).

RAKEL transforms the multi-label learning problem into an ensemble of *multi-class* classification problems. Specifically, each component learner in the ensemble is generated by considering a random subset of $\mathcal{Y}$. Let $\mathcal{S}_k \subset \mathcal{Y}$ denote a k-labelset which contains k random class labels in $\mathcal{Y}$. Accordingly, let $\sigma_{\mathcal{S}_k} : 2^{\mathcal{S}_k} \mapsto \mathcal{N}$ denote the injective function mapping from the power set of $\mathcal{S}_k$ to natural numbers. In view of $\mathcal{S}_k$, a corresponding multi-class training set can be constructed by shrinking the original label space $\mathcal{Y}$ into $\mathcal{S}_k$:

$$\mathcal{D}_{\mathcal{S}_k} = \left\{\left(\boldsymbol{x}_i, \sigma_{\mathcal{S}_k}(Y_i \cap \mathcal{S}_k)\right) \mid 1 \leq i \leq m\right\} \tag{7}$$

Here, the set of newly transformed labels in $\mathcal{D}_{\mathcal{S}_k}$ corresponds to $\Gamma_{\mathcal{S}_k} = \{\sigma_{\mathcal{S}_k}(Y_i \cap \mathcal{S}_k) \mid 1 \leq i \leq m\}$. As shown in Eq. (7), each instance $\boldsymbol{x}_i$ is transformed into a multi-class single-label example by mapping the intersection between Y_i and $\mathcal{S}_k$ into a new label in $\Gamma_{\mathcal{S}_k}$. After that, a multi-class classifier $g_{\mathcal{S}_k} : \mathcal{X} \mapsto \Gamma_{\mathcal{S}_k}$ can be induced for $\mathcal{S}_k$ by utilizing some multi-class learning algorithm $\mathcal{M}$, i.e., $g_{\mathcal{S}_k} \hookleftarrow \mathcal{M}(\mathcal{D}_{\mathcal{S}_k})$. To entirely explore the original label space $\mathcal{Y}$ with k-labelsets, an ensemble of n random k-labelsets $\mathcal{S}^{(r)}_k$ $(1 \leq r \leq n)$ can be created where each of them leads to a multi-class classifier $g_{\mathcal{S}^{(r)}_k}(\cdot)$. For unseen instance $\boldsymbol{x}$, its label set is predicted by referring to the following two quantities:

$$\tau(\boldsymbol{x}, y_j) = \sum_{r=1}^{n} \left[\!\!\left[y_j \in \mathcal{S}^{(r)}_k \right]\!\!\right]$$
$$\mu(\boldsymbol{x}, y_j) = \sum_{r=1}^{n} \left[\!\!\left[y_j \in \sigma^{-1}_{\mathcal{S}^{(r)}_k}\left(g_{\mathcal{S}^{(r)}_k}(\boldsymbol{x})\right) \right]\!\!\right] \tag{8}$$

Conceptually, $\tau(\boldsymbol{x}, y_j)$ counts the *maximum* number of votes that y_j can receive from the ensemble, whereas $\mu(\boldsymbol{x}, y_j)$ counts the *actual* number of votes that y_j does receive from the ensemble. Therefore, the predicted label set corresponds to: $Y = \{y_j \mid \mu(\boldsymbol{x}, y_j)/\tau(\boldsymbol{x}, y_j) > 0.5,\ 1 \leq j \leq q\}$. In general, CC assumes high-order label correlations (among subsets of labels) in its learning procedure and optimizes the evaluation measure of subset accuracy (measured w.r.t. k-labelset).

It is worth mentioning that many multi-label learning algorithms mainly work under the scenarios where the label space $\mathcal{Y}$ contains moderate number (tens or hundreds) of class labels. Nonetheless, in many applications the number of class labels in $\mathcal{Y}$ can be huge. For instance, a web page may be annotated with relevant labels from the pool of more than one million Wikipedia categories. In such case, the computational complexity of many multi-label learning algorithms might be prohibitively high. Even for binary decomposition, which is the simplest way to learn from multi-label data, building one independent classifier for each label is still too computational demanding given the huge number of class labels. Therefore, specific strategies need to be employed to handle huge number of labels. One feasible strategy is to find a low-dimensional embedding of the original label space by exploiting the sparsity of relevant labels, where the classification model is built within the embedded label space (Weston et al. 2011). Another strategy is to partition the original label space into different clusters based on tree structure, where the classification model is built within each leaf node (Agrawal et al. 2013).

Theory

Multi-label loss functions are usually *non-convex* and *discontinuous*, making them difficult to optimize directly. Therefore, in practice, most learning algorithms resort to optimizing (convex) *surrogate* loss functions. There are several theoretical studies about the *consistency* of surrogate loss functions, i.e., whether the expected risk of surrogate loss of a learner converges to the Bayes risk of multi-label loss as the training set size increases. Recently, a necessary and sufficient condition has been provided for the consistency of multi-label learning based on surrogate loss functions (Gao and Zhou 2011).

For *hamming loss*, the state-of-the-art multi-label learning approaches are proven to be inconsistent (Gao and Zhou 2011). For *ranking loss*, it has been shown that none pairwise convex surrogate loss defined on label pairs can be consistent; therefore, the *partial ranking loss* is introduced for multi-label learning, and some pairwise consistent surrogate loss function are provided (Gao and Zhou 2011). The univariate convex surrogate loss defined on single label can be consistent with partial ranking loss based on a reduction to the bipartite ranking problem (Dembczyński et al. 2012) although the reduction relaxes the original target.

Extensions

Multi-instance multi-label learning (MIML) (Zhou et al. 2012) tries to induce a function $h_{\text{MIML}} : 2^{\mathcal{X}} \mapsto 2^{\mathcal{Y}}$ from a training set $\{(X_i, Y_i) \mid 1 \leq i \leq m\}$, where $X_i \subseteq \mathcal{X}$ is a set of instances and Y_i is the set of class labels associated with X_i. The major difference between MIML and multi-label learning lies in the fact that each example in MIML is represented by a set of instances rather than a single instance. This framework is suitable to tasks involving complicated objects with inherent structures; e.g., a text document can be represented by a set of instances each corresponds to a section or paragraph. In addition to exploit the structural information for learning the predictor, MIML also offers the possibility of discovering the relation between semantic meanings and input patterns; e.g., it is possible to discover that the document owes a specific tag because of its several special paragraphs.

Superset label learning (SLL) (Liu and Dietterich 2012) tries to induce a function $h_{SLL} : \mathcal{X} \mapsto \mathcal{Y}$ from a training set $\{(\boldsymbol{x}_i, S_i) \mid 1 \leq i \leq m\}$, where $\boldsymbol{x}_i \in \mathcal{X}$ is an instance and $S_i \subseteq \mathcal{Y}$ is the set of candidate labels associated with $\boldsymbol{x}_i$ such that the (unknown) ground-truth label y_i belongs to S_i. The major difference between SLL and multi-label learning lies in the fact that each example in SLL is associated with multiple *candidate* labels among which only one label is valid. This framework is suitable to tasks where superset labeling information is readily available; e.g., a face in an image can be associated with all the names mentioned in the image's surrounding texts where only one name is valid.

Label distribution learning (LDL) (Geng et al. 2013) tries to induce a function $f_{\text{LDL}} : \mathcal{X} \mapsto \mathcal{P}(\mathcal{Y})$ from a training set $\{(\boldsymbol{x}_i, \mathcal{D}_i) \mid 1 \leq i \leq m\}$, where $\boldsymbol{x}_i$ is an instance and $\mathcal{D}_i = \{d_i^1, d_i^2, \cdots, d_i^q\}$ is the probability mass of the q labels associated with $\boldsymbol{x}_i$ such that $d_i^j \geq 0$ (1 $\leq$

$j \leq q$) and $\sum_{j=1}^{q} d_i^j = 1$. The major difference between LDL and multi-label learning lies in the fact that the associated labeling information for each example in LDL is real-valued probability mass rather than discrete-valued binary labels. This framework is suitable to tasks where the degree of labeling importance is inherently different; e.g., entities appearing in a natural scene have different importance in implying its scenic concepts.

Future Challenges

There are many research challenges to be addressed in the future. Firstly, label relations play a critical role in multi-label learning; however, there lacks principled mechanism for label relation exploitation. Secondly, it is generally difficult to get accurate and complete label annotations, particularly when each example has many labels. Thus, it is important to develop multi-label learning approaches that can learn from partially labeled data. Moreover, multi-label data usually suffers from inherent class imbalance and unequal misclassification costs; taking these properties into full consideration is desirable.

Recommended Reading

Agrawal R, Gupta A, Prabhu Y, Varma M (2013) Multi-label learning with millions of labels: recommending advertiser bid phrases for web pages. In: Proceedings of the 22nd international conference on world wide web, Rio de Janeiro, pp 13–24

Boutell MR, Luo J, Shen X, Brown CM (2004) Learning multi-label scene classification. Pattern Recognit 37(9):1757–1771

Dembczyński K, Kotłowski W, Hüllermeier E (2012) Consistent multilabel ranking through univariate loss minimization. In: Proceedings of the 29th international conference on machine learning, Edinburgh, pp 1319–1326

Elisseeff A, Weston J (2002) A kernel method for multi-labelled classification. In: Dietterich TG, Becker S, Ghahramani Z (eds) Advances in neural information processing systems, vol 14. MIT Press, Cambridge, pp 681–687

Gao W, Zhou Z-H (2011) On the consistency of multi-label learning. In: Proceedings of the 24th annual conference on learning theory, Budapest, pp 341–358

Geng X, Yin C, Zhou Z-H (2013) Facial age estimation by label distribution learning. IEEE Trans Pattern Anal Mach Intell 35(10):2401–2412

Liu L, Dietterich T (2012) A conditional multinomial mixture model for superset label learning. In: Bartlett P, Pereira FCN, Burges CJC, Bottou L, Weinberger KQ (eds) Advances in neural information processing systems, vol 25. MIT Press, Cambridge, pp 557–565

Read J, Pfahringer B, Holmes G, Frank E (2011) Classifier chains for multi-label classification. Mach Learn 85(3):333–359

Rousu J, Saunders C, Szedmak S, Shawe-Taylor J (2005) Learning hierarchical multi-category text classification models. In: Proceedings of the 22nd international conference on machine learning, Bonn, pp 774–751

Schapire RE, Singer Y (2000) Boostexter: a boosting-based system for text categorization. Mach Learn 39(2/3):135–168

Tsoumakas G, Katakis I, Vlahavas I (2011) Random k-labelsets for multi-label classification. IEEE Trans Knowl Data Eng 23(7):1079–1089

Weston J, Bengio S, Usunier N (2011) WSABIE: scaling up to large vocabulary image annotation. In: Proceedings of the 22nd international joint conference on artificial intelligence, Barcelona, pp 2764–2770

Zhang M-L, Zhou Z-H (2007) ML-kNN: a lazy learning approach to multi-label learning. Pattern Recognit 40(7):2038–2048

Zhang M-L, Zhou Z-H (2014) A review on multi-label learning algorithms. IEEE Trans Knowl Data Eng 26(8):1819–1837

Zhou Z-H, Zhang M-L, Huang S-J, Li Y-F (2012) Multi-instance multi-label learning. Artif Intell 176(1):2291–2320

Multi-objective Optimization

Synonyms

MOO; Multi-criteria optimization; Vector optimization

Definition

Multi-criteria optimization is concerned with the optimization of a vector of objectives, which can be the subject of a number of constraints or bounds. The goal of multi-objective optimization is usually to find or to approximate the set of Pareto-optimal solutions. A solution is Pareto-optimal if it cannot be improved in one objective without getting worse in another one.

Multiple Classifier Systems

▶ Ensemble Learning

Multiple-Instance Learning

Soumya Ray[1], Stephen Scott[2], and
Hendrik Blockeel[3,4]
[1]Case Western Reserve University, Cleveland,
OH,
USA
[2]University of Nebraska, Lincoln, NE, USA
[3]Katholieke Universiteit Leuven, Heverlee,
Leuven, Belgium
[4]Leiden Institute of Advanced Computer
Science, Heverlee, Belgium

Definition

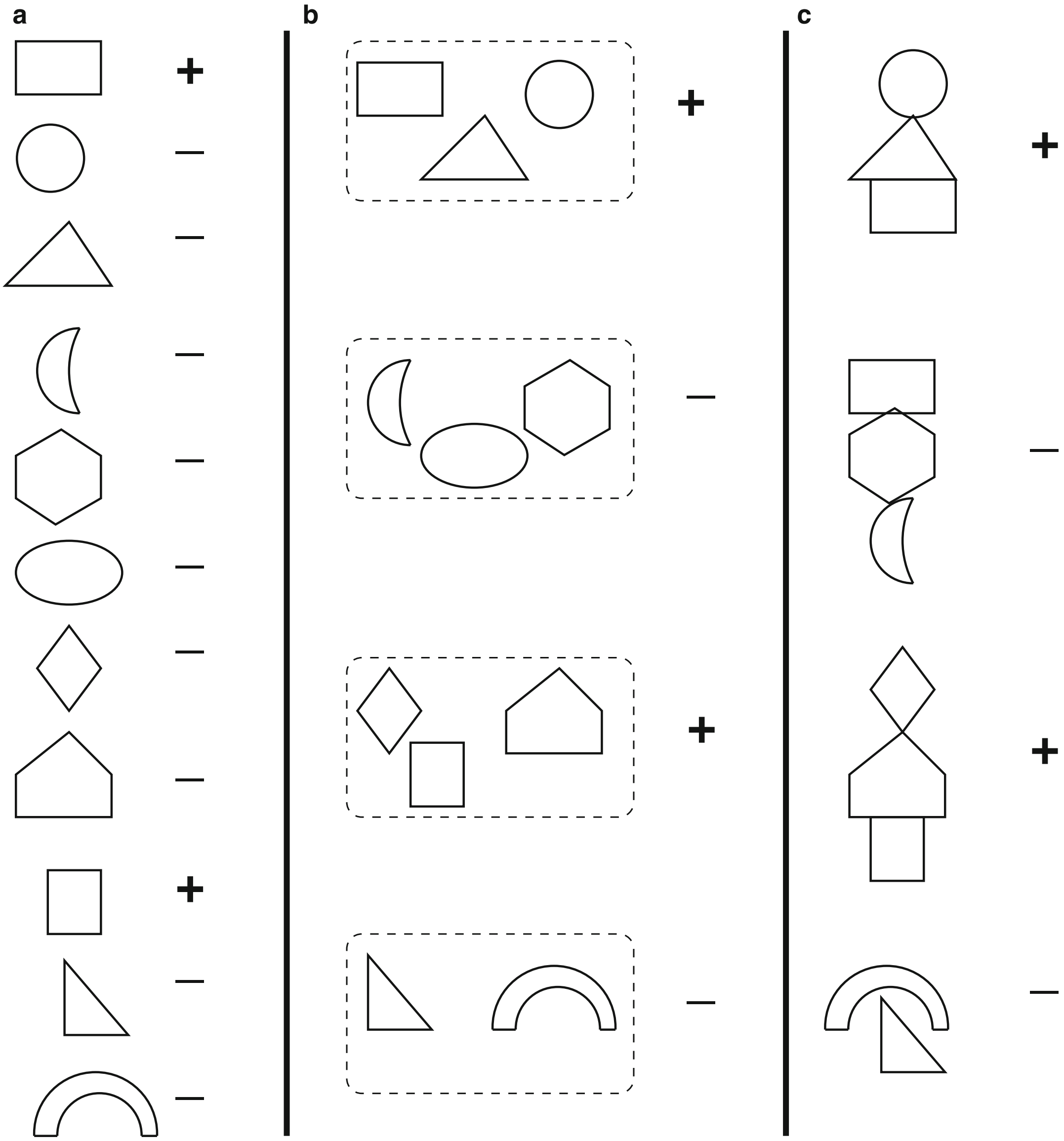
Multiple-instance (MI) learning is an extension of the standard supervised learning setting. In standard supervised learning, the input consists of a set of labeled instances each described by an attribute vector. The learner then induces a concept that relates the label of an instance to its attributes. In MI learning, the input consists of labeled examples (called “bags”) consisting of *multisets* of instances, each described by an attribute vector, and there are constraints that relate the label of each bag to the unknown labels of each instance. The MI learner then induces a concept that relates the label of a bag to the attributes describing the instances in it. This setting contains supervised learning as a special case: if each bag contains exactly one instance, it reduces to a standard supervised learning problem.

Motivation and Background

The MI setting was introduced by Dietterich et al. (1997) in the context of drug activity prediction. Drugs are typically molecules that fulfill some desired function by binding to a target. If we wish to learn the characteristics responsible for binding, a possible representation of the problem is to represent each molecule as a set of low-energy shapes or *conformations* and describe each conformation using a set of attributes. Each such bag of conformations is given a label corresponding to whether the molecule is active or inactive. To learn a classification model, an algorithm assumes that every instance in a bag labeled negative is actually negative, whereas at least one instance in a bag labeled positive is actually positive with respect to the underlying concept.

From a theoretical viewpoint, MI learning occupies an intermediate position between standard propositional supervised learning and first-order relational learning. Supervised learning is a special case of MI learning, while MI learning is a special case of first-order learning. It has been argued that the MI setting is a key transition between standard supervised and relational learning (DeRaedt 1998). At the same time, theoretical results exist that show that, under certain assumptions, certain concept classes that are probably approximately correct (PAC)-learnable (see PAC learning) in a supervised setting remain PAC-learnable in an MI setting. Thus, the MI setting is able to leverage some of the rich representational power of relational learners while not sacrificing the efficiency of propositional learners. Figure 1 illustrates the relationships between standard supervised learning, MI learning, and relational learning.

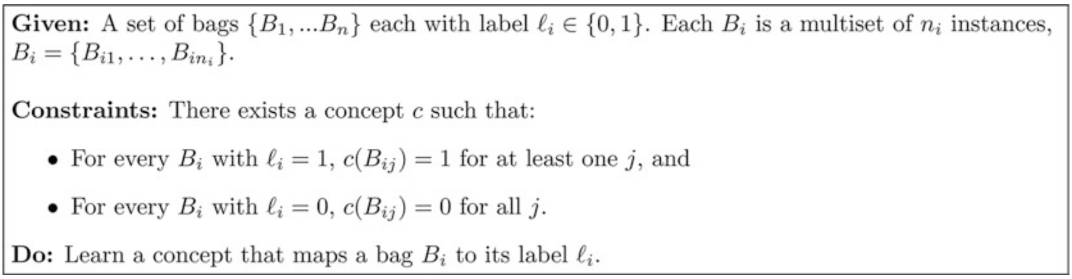
Since its introduction, a wide variety of tasks have been formulated as MI learning problems. Many new algorithms have been developed, and well-known supervised learning algorithms extended, to learn MI concepts. A great deal of work has also been done to understand what kinds of concepts can and cannot be learned efficiently in this setting. In the following sections, we discuss the theory, methods, and applications of MI learning in more detail.



Multiple-Instance Learning, Fig. 1 The relationship between supervised, multiple-instance (MI), and relational learning. (**a**) In supervised learning, each example (geometric figure) is labeled. A possible concept that explains the example labels shown is "the figure is a rectangle." (**b**) In MI learning, bags of examples are labeled. A possible concept that explains the bag labels shown is "the bag contains at least one figure that is a rectangle." (**c**) In relational learning, objects of arbitrary structure are labeled. A possible concept that explains the object labels shown is "the object is a stack of three figures and the bottom figure is a rectangle"

Structure of the Problem

The general MI classification task is shown in Fig. 2. The MI regression task is defined analogously by substituting a real-valued response for the classification label. In this case, the constraint used by the learning algorithm is that the response of any bag is equal to the response of at least one of the instances in it, for example, it could be equal to the largest response over all the instances.



Given: A set of bags $\{B_1, ...B_n\}$ each with label $\ell_i \in \{0,1\}$. Each B_i is a multiset of n_i instances, $B_i = \{B_{i1}, \ldots, B_{in_i}\}$.

Constraints: There exists a concept c such that:

- For every B_i with $\ell_i = 1$, $c(B_{ij}) = 1$ for at least one j, and
- For every B_i with $\ell_i = 0$, $c(B_{ij}) = 0$ for all j.

Do: Learn a concept that maps a bag B_i to its label ℓ_i.

Multiple-Instance Learning, Fig. 2 Statement of the multiple-instance classification problem

Notice the following problem characteristics:

- The number of instances in each bag can vary independently of other bags. This implies in particular that an MI algorithm must be able to handle bags with as few as one instance (this is a supervised learning setting) to bags with large numbers of instances.
- The number of instances in any positive bag that are "truly positive" could be many more than one – in fact, the definition does not rule out the case where *all* instances in a positive bag are "truly positive."
- The problem definition does not specify how the instances in any bag are related to each other.

Theory and Methods

In this section, we discuss some of the key algorithms and theoretical results in MI learning. We first discuss the methods and results for MI classification. Then we discuss the work on MI regression.

Multiple-Instance Classification

Axis-parallel rectangles (APRs) are a concept class that early work in MI classification focused on. These generative concepts specify upper and lower bounds for all numeric attributes describing each instance. An APR is said to "cover" an instance if the instance lies within it. An APR covers a bag if it covers at least one instance within it. The learning algorithm tries to find an APR such that it covers all positive bags and does not cover any negative bags.

An algorithm called "iterated discrimination" was proposed by Dietterich et al. (1997) to learn APRs from MI data. This algorithm has two phases. In the first phase, it iteratively chooses a set of "relevant" attributes and grows an APR using this set. This phase results in the construction of a very "tight" APR that covers just positive bags. In the second phase, the algorithm expands this APR so that with high probability, a new positive instance will fall within the APR. The key steps of the algorithm are outlined below. Note that initially, all attributes are considered to be "relevant."

The algorithm starts by choosing a random instance in a positive bag. Let us call this instance I_1. The smallest APR covering this instance is a point. The algorithm then expands this APR by finding the smallest APR that covers any instance from a yet uncovered positive bag; call the newly covered instance I_2. This process is continued, identifying new instances $I_3, \ldots, I_k$, until all positive bags are covered. At each step, the APR is "backfitted" in a way that is reminiscent of the later Expectation-Maximization (EM) approaches: each earlier choice is revisited, and I_j is replaced with an instance from the same bag that minimizes the current APR (which may or may not be the same as the one that minimized it at step j).

This process yields an APR that imposes maximally tight bounds on all attributes and covers all positive bags. Based on this APR, a new set of "relevant" attributes is selected as follows. An attribute's relevance is determined by how strongly it discriminates against negative instances, i.e., given the current APR bounds, how many negative instances the attribute excludes. Features are then chosen iteratively and greedily according to how relevant they are until all negative instances have been excluded. This yields a subset of (presumably relevant) attributes. The APR growth procedure in the previous paragraph is then repeated, with the size of an APR redefined as its size along relevant attributes only. The APR growth and attribute selection phases are repeated until the process converges.

The APR thus constructed may still be too tight, as it fits narrowly around the positive bags in the dataset. In the second phase of the algorithm, the APR bounds are further expanded using a kernel density estimate approach. Here, a probability distribution is constructed for each relevant attribute using Gaussian distributions centered at each instance in a positive bag. Then, the bounds on that attribute are adjusted so that with high probability, any positive instance will lie within the expanded APR.

Theoretical analyses of APR concepts have been performed along with the empirical approach, using Valiant's "probably approximately correct" (PAC)-learning model (Valiant 1984). In early work (Long and Tan 1998), it was shown that if each instance was drawn according to a fixed, unknown product distribution over the rational numbers, independently from every other instance, then an algorithm could PAC-learn APRs. Later, this result was improved in two ways (Auer et al. 1998). First, the restriction that the individual instances in each bag come from a product distribution was removed. Instead, each instance is generated by an arbitrary probability distribution (though each instance in a bag is still generated independently and identically distributed (iid) according to that one distribution). Second, the time and sample complexities for PAC-learning APRs were improved. Specifically, the algorithm described in this work PAC-learns APRs in

$$O\left(\frac{d^3n^2}{\epsilon^2}\log\frac{nd\log(1/\delta)}{\epsilon}\log\frac{d}{\delta}\right)$$

using

$$O\left(\frac{d^2n^2}{\epsilon^2}\log\frac{d}{\delta}\right)$$

time-labeled training bags. Here, d is the dimension of each instance, n is the (largest) number of instances per training bag, and ϵ and δ are parameters to the algorithm. A variant of this algorithm was empirically evaluated and found to be successful (Auer 1997).

Diverse Density (Maron 1998; Maron and Lozano-Pérez 1998) is a probabilistic generative framework for MI classification. The idea behind this framework is that, given a set of positive and negative bags, we wish to learn a concept that is "close" to at least one instance from each positive bag, while remaining "far" from every instance in every negative bag. Thus, the concept must describe a region of instance space that is "dense" in instances from positive bags and is also "diverse" in that it describes every positive bag. More formally, let

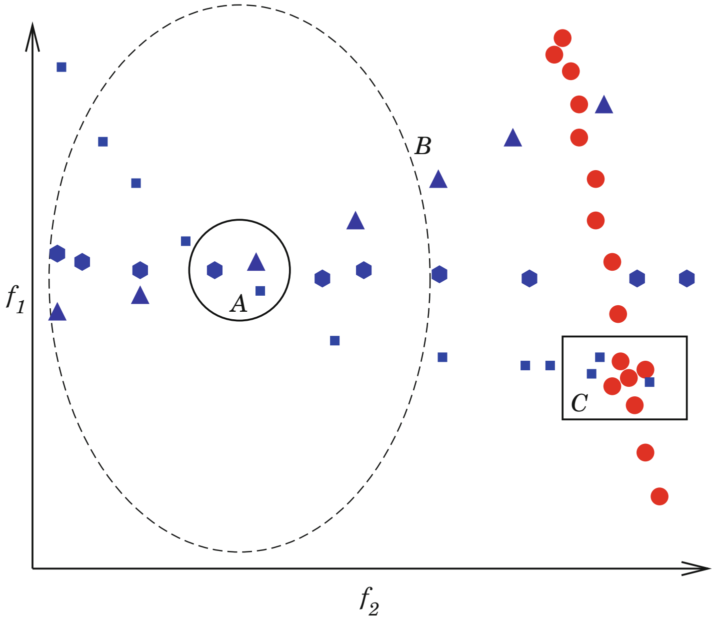
$$DD(t) = \frac{1}{Z}\left(\prod_i \Pr\left(t|B_i^+\right)\prod_i \Pr\left(t|B_i^-\right)\right),$$

where t is a candidate concept, B_i^+ represents the ith positive bag, and B_i^- represents the ith negative bag. We seek a concept that maximizes $DD(t)$. The concept generates the instances of a bag, rather than the bag itself. To score a concept with respect to a bag, we combine t's probabilities for instances using a function based on noisy-OR (Pearl 1998):

$$\Pr(t|B_i^+) \propto \left(1-\prod_j\left(1-\Pr\left(B_{ij}^+\in t\right)\right)\right) \tag{1}$$

$$\Pr\left(t|B_i^-\right) \propto \prod_j\left(1-\Pr\left(B_{ij}^-\in t\right)\right) \tag{2}$$

Here, the instances B_{ij}^+ and B_{ij}^- belonging to t are the "causes" of the "event" that "t is the target." The concept class investigated by Maron (1998) is the class of generative Gaussian models, which



Multiple-Instance Learning, Fig. 3 An illustration of the concept that Diverse Density searches for on a simple MI dataset with three positive bags and one negative bag, where each instance (represented by the geometric figures) is described by two attributes, f_1 and f_2. Each type of figure represents one bag, i.e., all *triangles* belong to one bag, all *circles* belong to a second bag, and so forth. The bag containing the *red circles* is negative, while the other bags are positive. Region C is a region of high density, because several instances belong to that region. Region A is a region of high "Diverse Density," because several instances *from different positive bags* belong to that region, and no instances from negative bags are nearby. Region B shows a concept that might be learned if the learning algorithm assumed that all instances in every positive bag are positive (Figure adapted from Maron and Lozano-Pérez (1998))

are parameterized by the mean μ and a "scale" $s = \frac{1}{2\sigma^2}$:

$$\Pr(B_{ij} \in t) \propto e^{-\sum_k \left(s_k (B_{ijk} - \mu_k)^2\right)},$$

where k ranges over attributes. Figure 3 illustrates a concept that Diverse Density might learn when applied to an MI dataset.

Diverse Density with k disjuncts is a variant of Diverse Density that has also been investigated (Maron 1998). This is a class of disjunctive Gaussian concepts, where the probability of an instance belonging to a concept is given by the maximum probability of belonging to any of the disjuncts.

EM-DD (Zhang and Goldman 2001) is an example of a class of algorithms that try to identify the "cause" of a bag's label using EM. These algorithms sometimes assume that there is a single instance in each bag that is responsible for the bag's label (though variants using "soft EM" are possible). The key idea behind this approach is as follows: from each positive bag, we take a random instance and assume that this instance is the relevant one. We learn a hypothesis from these relevant instances and all negative bags. Next, for each positive bag, we replace the current relevant instance by the instance most consistent with the learned hypothesis (which will initially not be the chosen instance in general). We then relearn the hypothesis with these new instances. This process is continued until the set of chosen instances does not change (or alternatively, the objective function of the classifier reaches a fixed point). This procedure has the advantage of being computationally efficient, since the learning algorithm only uses one instance from each positive bag. This approach has also been used in MI regression described later.

"Upgraded" supervised learning algorithms can be used in an MI setting by suitably modifying their objective functions. Below, we summarize some of the algorithms that have been derived in this way.

1. ▸ *Decision Tree induction* algorithms have been adapted to the MI setting (Blockeel et al. 2005). The standard algorithm measures the quality of a split on an attribute by considering the class label distribution in the child nodes produced. In the MI case, this distribution is uncertain, because the true instance labels in positive bags are unknown. However, some rules have been identified that lead to empirically good MI trees: (1) use an asymmetric heuristic that favors early creation of pure positive (rather than negative) leaves; (2) once a positive leaf has been created, remove all other instances of the bags covered by this leaf; (3) abandon the depth-first or breadth-first order in which nodes are usually split, adopting a best-first strategy instead (indeed, because of (2), the result of tree learning is now sensitive to the order in which the nodes are split).
2. ▸ *Artificial Neural Networks* have been adapted to the MI setting by representing the bag classifier as a network that combines several copies of a smaller network, which represents the instance classifier, with a smooth approximation of the *max* combining function (Ramon and DeRaedt 2000). Weight update rules for a backpropagation algorithm working on this network have been derived. Later work on MI neural networks has been performed independently by others (Zhou and Zhang 2002).
3. ▸ *Logistic Regression* has been adapted to the MI setting by using it as an instance-based classifier and combining the instance-level probabilities using functions like softmax (Ray and Craven 2005) and arithmetic and geometric averages (Xu and Frank 2004).
4. The ▸ *k-Nearest Neighbor* algorithm has been adapted to the MI setting by using set-based distance metrics, such as variants based on the Hausdorff distance. However, this alone does not solve the problem – it is possible for a positive bag to be mistakenly classified negative if it contains a "true negative" instance that happens to be much closer to negative instances in other negative bags. To solve this, a "Citation-kNN" (Wang and Zucker 2000) approach has been proposed that also considers, for each bag B, the labels of those bags for which B is a nearest neighbor.
5. ▸ *Support Vector Machines* have been adapted to the MI setting in several ways. In one method, the constraints in the quadratic program for SVMs is modified to account for the fact that certain instance labels are unknown but have constraints relating them (Andrews et al. 2003). In another method, new kernels are designed for MI data by modifying standard supervised SVM kernels (Gartner et al. 2002) or designing new kernels (Tao et al. 2004). The modification allows these MI kernels to distinguish between positive and negative bags if the supervised kernel could distinguish between ("true") positive and negative instances.
6. ▸ *Rule learning algorithms* have been adapted to the MI setting in two ways. One method has investigated upgrading a supervised rule-learner, the RIPPER system (Cohen 1995), to the MI setting by modifying its objective function to account for bags and addressing several issues that resulted. Another method has investigated using general-purpose relational algorithms, such as FOIL (Quinlan 1990) and TILDE (Blockeel and De Raedt 1998), and providing them with an appropriate ▸ inductive bias so that they learn the MI concepts. Further, it has been observed that techniques from MI learning can also be used inside relational learning algorithms (Alphonse and Matwin 2002).

A large-scale empirical analysis of several such propositional supervised learning algorithms and their MI counterparts has been performed (Ray and Craven 2005). This analysis concludes that (1) no single MI algorithm works well across all problems (thus, different inductive biases are suited to different problems), (2) some MI algorithms consistently perform better than their supervised counterparts but others do not (hence, for these biases, there seems room for improvement), and (3) assigning a larger weight to false positives than to false negatives is a simple but effective method to adapt supervised learning algorithms to the MI setting. It was also observed that the advantages of MI learners may be more pronounced if they would be evaluated on the task of labeling individual instances rather than bags.

Along with "upgrading" supervised learning algorithms, a *theoretical analysis of supervised learners* learning with MI data has been carried out (Blum and Kalai 1998). In particular, the MI problem has been related to the problem of learning in the presence of classification noise (i.e., each training example's label is flipped with some probability $<1/2$). This implies that any concept class that is PAC-learnable in the presence of such noise is also learnable in the MI learning model when each instance of a bag is drawn iid. Since many concept classes are learnable under this noise assumption (using, e.g., *statistical queries* Kearns 1998), Blum and Kalai's result implies PAC learnability of many concept classes. Further, they improved on previous learnability results (Auer et al. 1998) by reducing the number of training bags required for PAC learning by about a factor of n with only an increase in time complexity of about $\log n/\epsilon$.

Besides these positive results, a *negative learnability result* describing when it is hard to learn concepts from MI data is also known (Auer et al. 1998). Specifically, if the instances of each bag are allowed collectively to be generated according to an arbitrary distribution, learning from MI examples is as hard as PAC-learning disjunctive normal form (DNF) formulas from single-instance examples, which is an open problem in learning theory that is believed to be hard. Further, it has been shown that if an efficient algorithm exists for the non-iid case that outputs as its hypothesis an axis-parallel rectangle, then NP = RP (Randomized Polynomial time; see, e.g., Papadimitriou 1994), which is very unlikely.

Learning from structured MI data has received some attention (McGovern and Jensen 2003). In this work, each instance is a graph, and a bag is a set of graphs (e.g., a bag could consist of certain subgraphs of a larger graph). To learn the concepts in this structured space, the authors use a modified form of the Diverse Density algorithm discussed above. As before, the concept being searched for is a point (which corresponds to a graph in this case). The main modification is the use of the size of the maximal common subgraph to estimate the probability of a concept – i.e., the probability of a concept given a bag is estimated as proportional to the size of the maximal common subgraph between the concept and any instance in the bag.

Multiple-Instance Regression

Regression problems in an MI setting have received less attention than the classification problem. Two key directions have been explored in this setting. One direction extends the well-known standard ▶ linear regression method to the MI setting. The other direction considers extending various MI classification methods to a regression setting.

In *MI linear regression* (Ray and Page 2001) (referred to as multiple-instance regression in the cited work), it is assumed that the hypothesis underlying the data is a linear model with Gaussian noise on the value of the dependent variable (which is the response). Further, it is assumed that it is sufficient to model one instance from each bag, i.e., that there is some *primary* instance which is responsible for the real-valued label. Ideally, one would like to find a hyperplane that minimizes the squared error with respect to these primary instances. However, these instances are unknown during training. The authors conjecture that, given enough data, a good approximation to the ideal is given by the "best-fit" hyperplane, defined as the hyperplane that minimizes the training set squared error by fitting one instance from each bag such that the response of the fitted instance most closely matches the bag response. This conjecture will be true if the nonprimary instances are not a better fit to a hyperplane than the primary instances. However, exactly finding the "best-fit" hyperplane is intractable. It is shown that the decision problem "Is there a hyperplane which perfectly fits one instance from each bag?" is NP-complete for arbitrary numbers of bags, attributes, and at most three instances per bag. Thus, the authors propose an approximation algorithm which iterates between choosing instances and learning linear regression models that best fit them, similar to the EM-DD algorithm described earlier.

Another direction has explored *extending MI classification algorithms* to the regression setting. This approach (Dooly et al. 2002) uses algorithms like Citation-kNN and Diverse Density to learn real-valued concepts. To predict a real value, the approach uses the average of the nearest neighbor responses or interprets the Gaussian "probability" as a real number for Diverse Density.

Recent work has analyzed the Diverse Density-based regression in the *online* model (Angluin 1988; Littlestone 1988) (see ▸ Online Learning). In the online model, learning proceeds in *trials*, where in each trial a single example is selected adversarially and given to the learner for classification. After the learner predicts a label, the true label is revealed and the learner incurs a *loss* based on whether its prediction was correct. The goal of the online learner is to minimize the loss over all trials. Online learning is harder than PAC learning in that there are some PAC-learnable concept classes that are not online learnable.

In the regression setting above (Dooly et al. 2006), there is a point concept, and the label of each bag is a function of the distance between the concept and the point in the bag closest to the target. It is shown that similar to Auer et al.'s lower bound, learning in this setting using labeled bags alone is as hard as learning DNF. They then define an *MI membership query* (MI-MQ) in which an adversary defines a bag $B = \{p_1, \ldots, p_n\}$ and the learner is allowed to ask an oracle for the label of bag $B + \vec{v} = \{p_1 + \vec{v}, \ldots, p_n + \vec{v}\}$ for any d-dimensional vector $\vec{v}$. Their algorithm then uses this MI-MQ oracle to online learn a real-valued MI concept in time $O(dn^2)$.

Applications

In this section, we describe domains where MI learning problems have been formulated.

Drug activity was the motivating application for the MI representation (Dietterich et al. 1997). Drugs are typically molecules that fulfill some desired function by binding to a target. In this domain, we wish to predict how strongly a given molecule will bind to a target. Each molecule is a three-dimensional entity and takes on multiple shapes or *conformations* in solution. We know that for every molecule showing activity, at least one of its low-energy conformations possesses the right shape for interacting with the target. Similarly, if the molecule does not show drug-like activity, none of its conformations possess the right shape for interaction. Thus, each molecule is represented as a bag, where each instance is a low-energy conformation of the molecule. A well-known example from this domain is the MUSK dataset. The positive class in this data consists of molecules that smell "musky." This dataset has two variants, MUSK1 and MUSK2, both with similar numbers of bags, with MUSK2 having many more instances per bag.

Content-Based Image Retrieval is another domain where the MI representation has been used (Maron and Lozano-Pérez 1998; Zhang et al. 2002). In this domain, the task is to find images that contain objects of interest, such as tigers, in a database of images. An image is represented by a bag. An instance in a bag corresponds to a segment in the image, obtained by some segmentation technique. The underlying assumption is that the object of interest is contained in (at least) one segment of the image. For example, if we are trying to find images of mountains in a database, it is reasonable to expect most images of mountains to have certain distinctive segments characteristic of mountains. An MI learning algorithm should be able to use the segmented images to learn a concept that represents the shape of a mountain and use the learned concept to collect images of mountains from the database.

The *identification of protein families* has been framed as an MI problem (Tao et al. 2004). The objective in that work is to classify given protein sequences according to whether they belong to the family of thioredoxin-fold proteins. The given proteins are first aligned with respect to a motif that is known to be conserved in the members of the family. Each aligned protein is represented by a bag. A bag is labeled positive if the protein belongs to the family, and negative otherwise. An instance in a bag corresponds to a position in a fixed length sequence around the conserved motif. Each position is described by a vector of attributes; each attribute describes the properties of the amino acid at that position and is smoothed using the same properties from its neighbors.

Text Categorization is another domain that has used the MI representation (Andrews et al. 2003; Ray and Craven 2005). In this domain, the task is to classify a document as belonging to a certain category or not. Often, whether the document belongs to the specified category is the function of a few passages in the document. These passages are however not labeled with the category information. Thus, a document could be represented as a set of passages. We assume that each positive document (i.e., that belongs to the specified category) has at least one passage that contains words that indicate category membership. On the other hand, a negative document (that does not belong to the category) has no passage that contains words indicating category membership. This formulation has been used to classify whether MEDLINE documents should be annotated with specific MeSH terms (Andrews et al.) and to determine if specific documents should be annotated with terms from the Gene Ontology (Ray and Craven 2005).

Time-series data from the hard drives have been used to define an MI problem (Murray et al. 2005). The task here is to distinguish drives that fail from others. Each hard drive is a bag. Each instance in the bag is a fixed-size window over timepoints when the drive's state was measured using certain attributes. In the training set, each drive is labeled according to whether it failed during a window of observation. An interesting aspect to prediction in this setting is that it is done online, i.e., the algorithm learns a classifier for instances, which is applied to each instance as it becomes available in time. The authors learn a naïve Bayes model using an EM-based approach to solve this problem.

Discovering useful subgoals in reinforcement learning has been formulated as an MI problem (McGovern and Barto 2001). Imagine that a robot has to get from one room to another by passing through a connecting door. If the robot knew of the existence of the door, it could decompose the problem into two simpler subproblems to be solved separately: getting from the initial location in the first room to the door and then getting from the door to its destination. How could the robot discover such a "useful subgoal?" One approach formulates this as an MI problem. Each trajectory of the robot, where the robot starts at the source and then moves for some number of time steps, is considered to be a bag. An instance in a bag is a state of the world, which records observations such as "is the robot's current location a door?" Trajectories that reach the destination are positive, while those that do not are negative. Given this data, we can learn a classifier that predicts which states are more likely to be seen on successful trajectories than on unsuccessful ones. These states are taken to be useful subgoals. In the previous example, the MI algorithm could learn that the state "location is a door" is a useful subgoal, since it appears on all successful trajectories, but infrequently on unsuccessful ones.

Future Directions

MI learning remains an active research area. One direction that is being explored relaxes the "constraints" in Fig. 2 in different ways (Tao et al. 2004; Weidmann et al. 2003). For example, one could consider constraints where at least a certain number (or fraction) of instances have to be positive for a bag to be labeled positive. Similarly, it may be the case that a bag is labeled positive only if it does not contain a specific instance. Such relaxations are often studied as "generalized multiple-instance learning."

One such generalization of MI learning has been formally studied under the name "geometric patterns." In this setting, the target concept consists of a collection of APRs, and a bag is labeled positive if and only if (1) each of its points lies in a target APR and (2) every target APR contains a point. Noise-tolerant PAC algorithms (Goldman and Scott 1999) and online algorithms (Goldman et al. 2001) have been presented for such concept classes. These algorithms make no assumptions on the distribution used to generate the bags (e.g., instances might not be generated by an iid process). This does not violate Auer et al.'s lower bound since these algorithms do not scale with the dimension of the input space.

Another recent direction explores the connections between MI and semi-supervised learnings. Semi-supervised learning generally refers to learning from a setting where some instance labels are unknown. MI learning can be viewed as one example of this setting. Exploiting this connection between MI learning and other methods for semi-supervised learning, recent work (Rahmani and Goldman 2006) proposes an approach where an MI problem is transformed into a semi-supervised learning problem. An advantage of the approach is that it automatically also takes into account unlabeled bags.

Cross-References

- ▶ Artificial Neural Network
- ▶ Attribute
- ▶ Classification
- ▶ Data Set
- ▶ Decision Tree
- ▶ Expectation Maximization Clustering
- ▶ First-Order Logic
- ▶ Gaussian Distribution
- ▶ Inductive Logic Programming
- ▶ Kernel Methods
- ▶ Linear Regression
- ▶ Multi-Instance Learning
- ▶ Nearest Neighbor
- ▶ Noise
- ▶ Online Learning
- ▶ PAC Learning
- ▶ Relational Learning
- ▶ Supervised Learning

Recommended Reading

Alphonse E, Matwin S (2002) Feature subset selection and inductive logic programming. In: Proceedings of the 19th international conference on machine learning, Sydney. Morgan Kaufmann, San Francisco, pp 11–18

Andrews S, Tsochantaridis I, Hofmann T (2003) Support vector machines for multiple-instance learning. In: Becker S, Thrun S, Obermayer K (eds) Advances in neural information processing systems, vol 15. MIT, Cambridge, pp 561–568

Angluin D (1988) Queries and concept learning. Mach Learn 2(4):319–342

Auer P (1997) On learning from multi-instance examples: empirical evaluation of a theoretical approach. In: Proceedings of the 14th international conference on machine learning, Nashville. Morgan Kaufmann, San Francisco, pp 21–29

Auer P, Long PM, Srinivasan A (1998) Approximating hyper-rectangles: learning and pseudorandom sets. J Comput Syst Sci 57(3):376–388

Blockeel H, De Raedt L (1998) Top-down induction of first order logical decision trees. Artif Intell 101(1–2):285–297

Blockeel H, Page D, Srinivasan A (2005) Multi-instance tree learning. In: Proceedings of 22nd international conference on machine learning, Bonn, pp 57–64

Blum A, Kalai A (1998) A note on learning from multiple-instance examples. Mach Learn J 30(1):23–29

Cohen WW (1995) Fast effective rule induction. In: Proceedings of the 12th international conference on machine learning, Tahoe City. Morgan Kaufmann, San Francisco

DeRaedt L (1998) Attribute-value learning versus inductive logic programming: the missing links. In: Proceedings of the eighth international conference on inductive logic programming, Madison. Springer, New York, pp 1–8

Dietterich T, Lathrop R, Lozano-Perez T (1997) Solving the multiple-instance problem with axis-parallel rectangles. Artif Intell 89(1–2):31–71

Dooly DR, Goldman SA, Kwek SS (2006) Real-valued multiple-instance learning with queries. J Comput Syst Sci 72(1):1–15

Dooly DR, Zhang Q, Goldman SA, Amar RA (2002) Multiple-instance learning of real-valued data. J Mach Learn Res 3:651–678

Gartner T, Flach PA, Kowalczyk A, Smola AJ (2002) Multi-instance kernels. In: Sammut C, Hoffmann A (eds) Proceedings of the 19th international conference on machine learning, Sydney. Morgan Kaufmann, San Francisco, pp 179–186

Goldman SA, Kwek SK, Scott SD (2001) Agnostic learning of geometric patterns. J Comput Syst Sci 6(1):123–151

Goldman SA, Scott SD (1999) A theoretical and empirical study of a noise-tolerant algorithm to learn geometric patterns. Mach Learn 37(1):5–49

Kearns M (1998) Efficient noise-tolerant learning from statistical queries. J ACM 45(6):983–1006

Long PM, Tan L (1998) PAC learning axis-aligned rectangles with respect to product distributions from multiple-instance examples. Mach Learn 30(1):7–21

Littlestone N (1988) Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. Mach Learn 2(4):285–318

Maron O (1998) Learning from ambiguity. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge

Maron O, Lozano-Pérez T (1998) A framework for multiple-instance learning. In: Jordan MI, Kearns MJ, Solla SA (eds) Advances in neural information processing systems, Denver, vol 10. MIT, Cambridge, pp 570–576

McGovern A, Barto AG (2001) Automatic discovery of subgoals in reinforcement learning using diverse density. In: Proceedings of the 18th international conference on machine learning, Williamstown. Morgan Kaufmann, San Francisco, pp 361–368

McGovern A, Jensen D (2003) Identifying predictive structures in relational data using multiple instance learning. In: Proceedings of the 20th international conference on machine learning, Washington, DC. AAAI, Menlo Park, pp 528–535

Murray JF, Hughes GF, Kreutz-Delgado K (2005) Machine learning methods for predicting failures in hard drives: a multiple-instance application. J Mach Learn Res 6:783–816

Papadimitriou C (1994) Computational complexity. Addison-Wesley, Boston

Pearl J (1998) Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann, San Mateo

Quinlan JR (1990) Learning logical definitions from relations. Mach Learn 5:239–266

Rahmani R, Goldman SA (2006) MISSL: multiple-instance semi-supervised learning. In: Proceedings of the 23rd international conference on machine learning, Pittsburgh. ACM, New York, pp 705–712

Ramon J, DeRaedt L (2000) Multi instance neural networks. In: Proceedings of ICML-2000 workshop on attribute-value and relational learning

Ray S, Craven M (2005) Supervised versus multiple-instance learning: an empirical comparison. In: Proceedings of the 22nd international conference on machine learning, Bonn. ACM, New York, pp 697–704

Ray S, Page D (2001) Multiple instance regression. In: Proceedings of the 18th international conference on machine learning, Williamstown. Morgan Kaufmann

Tao Q, Scott SD, Vinodchandran NV (2004) SVM-based generalized multiple-instance learning via approximate box counting. In: Proceedings of the 21st international conference on machine learning, Banff. Morgan Kaufmann, San Francisco, pp 779–806

Valiant LG (1984) A theory of the learnable. Commun ACM 27(11):1134–1142

Wang J, Zucker JD (2000) Solving the multiple-instance problem: a lazy learning approach. In: Proceedings of the 17th international conference on machine learning, Stanford. Morgan Kaufmann, San Francisco, pp 1119–1125

Weidmann N, Frank E, Pfahringer B (2003) A two-level learning method for generalized multi-instance problems. In: Proceedings of the European conference on machine learning, Cavtat-Dubrovnik. Springer, Berlin/Heidelberg, pp 468–479

Xu X, Frank E (2004) Logistic regression and boosting for labeled bags of instances. In: Proceedings of the Pacific-Asia conference on knowledge discovery and data mining, Sydney, pp 272–281

Zhang Q, Goldman S (2001) EM-DD: an improved multiple-instance learning technique. In: Advances in neural information processing systems, Vancouver. MIT, pp 1073–1080

Zhang Q, Yu W, Goldman S, Fritts J (2002) Content-based image retrieval using multiple-instance learning. In: Proceedings of the 19th international conference on machine learning, Sydney. Morgan Kaufmann, San Francisco, pp 682–689

Zhou ZH, Zhang ML (2002) Neural networks for multi-instance learning. Technical report, Nanjing University, Nanjing

Multi-relational Data Mining

Luc De Raedt
Department of Computer Science, Katholieke Universiteit Leuven, Heverlee, Leuven, Belgium

Synonyms

Inductive logic programming; Relational learning; Statistical relational learning

Definition

Multi-relational data mining is the subfield of knowledge discovery that is concerned with the mining of multiple tables or relations in a database. This allows it to cope with structured data in the form of complex data that cannot easily be represented using a single table, or an ▶ attribute as is common in machine learning.

Relevant techniques of multi-relational data mining include those from relational learning, statistical relational learning, and inductive logic programming.

Cross-References

▶ Inductive Logic Programming

Recommended Reading

Dzeroski S, Lavrac N (eds) (2001) Relational data mining. Springer, Berlin

Multistrategy Ensemble Learning

Definition

Every ▶ ensemble learning strategy might be expected to have unique effects on the base learner. Combining multiple ensemble learning algorithms might hence be expected to provide benefit. For example, ▶ Multi-Boosting combines ▶ AdaBoost and a variant of ▶ Bagging, obtaining most of AdaBoost's ▶ bias reduction coupled with most of Bagging's ▶ variance reduction. Similarly, ▶ Random Forests combines Bagging's variance reduction with ▶ Random Subspaces' bias reduction.

Cross-References

▶ Ensemble Learning
▶ MultiBoosting
▶ Random Forests

Recommended Reading

Webb GI, Zheng Z (2004) Multistrategy ensemble learning: reducing error by combining ensemble learning techniques. IEEE Trans Knowl Data Eng 16(8): 980–991

Multitask Learning

▶ Inductive Transfer

Must-Link Constraint

A pairwise constraint between two items indicating that they should be placed into the same cluster in the final partition.

N

Naïve Bayes

Geoffrey I. Webb
Faculty of Information Technology, Monash University, Victoria, Australia

Synonyms

Idiot's Bayes; Simple Bayes

Definition

Naïve Bayes is a simple learning algorithm that utilizes ▶ Bayes' rule together with a strong assumption that the attributes are conditionally independent given the class. While this independence assumption is often violated in practice, naïve Bayes nonetheless often delivers competitive classification accuracy. Coupled with its computational efficiency and many other desirable features, this leads to naïve Bayes being widely applied in practice.

Motivation and Background

Naïve Bayes provides a mechanism for using the information in sample data to estimate the posterior probability $\mathrm{P}(y|\mathbf{x})$ of each class y given an object $\mathbf{x}$. Once we have such estimates, we can use them for ▶ classification or other decision support applications.

Naïve Bayes' features include the following:

- *Computational efficiency*: ▶ training time is linear with respect to both the number of ▶ training examples and the number of ▶ attributes, and ▶ classification time is linear with respect to the number of attributes and unaffected by the number of training examples.
- *Low variance*: because naïve Bayes does not directly fit the posterior distribution, it has low variance, albeit at the cost of high ▶ bias.
- *Incremental learning*: naïve Bayes operates from estimates of low-order probabilities that are derived from the training data. These can readily be updated as new training data are acquired.
- *Direct prediction of posterior probabilities.*
- *Robustness in the face of noise*: naïve Bayes always uses all attributes for all predictions and hence is relatively insensitive to ▶ noise in the examples to be classified. Because it uses probabilities, it is also relatively insensitive to noise in the training data.
- *Robustness in the face of missing values*: because naïve Bayes always uses all attributes for all predictions, if one attribute value is missing, information from other attributes is still used, resulting in graceful degradation in performance. It is also relatively insensitive to

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

missing attribute values in the training data due to its probabilistic framework.

Structure of Learning System

Naïve Bayes is a form of Bayesian network classifier based on ▶ Bayes' rule:

$$P(y|\mathbf{x}) = P(y)P(\mathbf{x}|y)/P(\mathbf{x}) \qquad (1)$$

together with an assumption that the attributes are conditionally independent given the class. For ▶ attribute-value data, this assumption entitles

$$P(\mathbf{x}|y) = \prod_{i=1}^{n} P(x_i|y) \qquad (2)$$

where x_i is the value of the ith attribute in $\mathbf{x}$ and n is the number of attributes:

$$P(\mathbf{x}) = \prod_{i=1}^{k} P(c_i)P(\mathbf{x}|c_i) \qquad (3)$$

where k is the number of classes and c_i is the ith class. Thus, (1) can be calculated by normalizing the numerators of the right-hand side of the equation.

The resulting classifier uses a linear model, equivalent to that used by ▶ logistic regression, differing only in the manner in which the parameters are chosen.

For ▶ categorical attributes, the required probabilities $P(y)$ and $P(x_i|y)$ are normally derived from frequency counts stored in arrays whose values are calculated by a single pass through the training data at training time. These arrays can be updated as new data are acquired, supporting ▶ incremental learning. Probability estimates are usually derived from the frequency counts using smoothing functions such as the ▶ Laplace estimate or an m-estimate.

For ▶ numeric attributes, either the data are discretized (see ▶ discretization) or probability density estimation is employed.

In ▶ text mining, two variants of naïve Bayes are often employed (McCallum and Nigam 1998). The *multivariate Bernoulli model* utilizes naïve Bayes as described above, with each document represented as a vector of binary variables, each representing the presence or absence of a specific word. However, only the words that are present in a document are considered when calculating the probabilities for that document.

In contrast, the *multinomial model* uses information about the number of times a word appears in a document. It treats each occurrence of a word in a document as a separate event. These events are assumed independent of each other. Hence the probability of a document given a class is the product of the probabilities of each word event given the class.

Cross-References

▶ Bayesian Methods
▶ Semi-naive Bayesian Learning

Recommended Reading

Lewis D (1998) Naive Bayes at forty: the independence assumption in information retrieval. In: Proceedings of the 10th European conference on machine learning (ECML-98), Chemnitz. Springer, Berlin, pp 4–15

McCallum A, Nigam K (1998) A comparison of event models for Naive Bayes text classification. In: AAAI-98 workshop on learning for text categorization. AAAI Press, Menlo Park, pp 41–48

NCL

▶ Negative Correlation Learning

NC-Learning

▶ Negative Correlation Learning

Nearest Neighbor

Eamonn Keogh
University of California-Riverside, Riverside,
CA, USA

Synonyms

Closest point; Most similar point

Definition

In a data collection M, the *nearest neighbor* to a data object q is the data object M_i, which minimizes dist (q, M_i), where dist is a *distance measure* defined for the objects in question. Note that the fact that the object M_i is the nearest neighbor to q does not imply that q is the nearest neighbor to M_i.

Motivation and Background

Nearest neighbors are useful in many machine learning and data mining tasks, such as ▶ classification, ▶ anomaly detection and motif discovery and in more general tasks such as spell checking, vector quantization, plagiarism detection, web search, and recommender systems.

The naive method to find the nearest neighbor to a point q requires a linear scan of all objects in M. Since this may be unacceptably slow for large datasets and/or computationally demanding distance measures, there is a huge amount of literature on speeding up nearest neighbor searches (query-by-content). The fastest methods depend on the distance measure used, whether the data is disk resident or in main memory, and the structure of the data itself. Many methods are based on the R-tree (Guttman 1984) or one of its variants (Manolopoulos et al. 2005). However, in recent years there has been an increased awareness that for many applications approximate nearest neighbors may suffice. This has led to the development of techniques like *locality sensitive hashing*, which finds high-quality approximate nearest neighbors in constant time.

The definition of nearest neighbor allows for the definition of one of the simplest classification schemes, the *nearest neighbor classifier*.

The major database (SIGMOD, VLDB, and PODS) and data mining (SIGKDD, ICDM, and SDM) conferences typically feature several papers on novel distance measures and techniques for speeding up nearest neighbor search. Pavel et al.'s book provides an excellent overview on the state-of-the-art techniques in nearest neighbor searching.

Recommended Reading

Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD international conference on management of data. ACM, New York, pp 47–57. ISBN: 0-89791-128-8

Manolopoulos Y, Nanopoulos A, Papadopoulos AN, Theodoridis Y (2005) R-trees: theory and applications. Springer, Berlin

Zezula P, Amato G, Dohnal V, Batko M (2005) Similarity search: the metric space approach. In: Advances in database systems, vol 32. Springer, New York, p 220. ISBN:0-387-29146-6

Nearest Neighbor Methods

▶ Instance-Based Learning

Negative Correlation Learning

Synonyms

NC-learning; NCL

Definition

Negative correlation learning (Liu and Yao 1999) is an ▶ ensemble learning technique. It can be used for regression or classification problems, though with classification problems the models must be capable of producing posterior probabilities. The model outputs are combined with a uniformly weighted average. The squared error is augmented with a penalty term which takes into account the diversity of the ensemble. The error for the ith model is,

$$E(f_i(x)) = \frac{1}{2}(f_i(x) - d)^2 - \lambda(f_i(x) - \bar{f}(x))^2. \qquad (1)$$

The coefficient λ determines the balance between optimizing individual accuracy, and optimizing ensemble diversity. With $\lambda = 0$, the models are trained independently, with no emphasis on diversity. With $\lambda = 1$, the models are tightly coupled, and the ensemble is trained as a single unit. Theoretical studies (Brown et al. 2006) have shown that NC works by directly optimizing the ▸ bias-variance-covariance trade-off, thus it explicitly *manages* the ensemble diversity. When the complexity of the individuals is sufficient to have high individual accuracy, NC provides little benefit. When the complexity is low, NC with a well-chosen λ can provide significant performance improvements. Thus the best situation to make use of the NC framework is with a large number of low accuracy models.

Recommended Reading

Brown G, Wyatt JL, Tino P (2006) Managing diversity in regression ensembles. J Mach Learn Res 6:1621–1650

Liu Y, Yao X (1999) Ensemble learning via negative correlation. Neural Netw 12(10):1399–1404

Negative Predictive Value

Negative Predictive Value (NPV) is defined as a ratio of true negatives to the total number of negatives predicted by a model. This is defined with reference to a special case of the ▸ confusion matrix with two classes – one designated the *positive* class and the other the *negative* class – as indicated in Table 1.

Negative Predictive Value, Table 1 The outcomes of classification into positive and negative classes

		Assigned class	
		Positive	Negative
Actual class	Positive	True positive (TP)	False negative (FN)
	Negative	False positive (FP)	True negative (TN)

NPV can then be defined in terms of true negatives and false negatives as follows.

$$\text{NPV} = \text{TN}/(\text{TN} + \text{FN})$$

Net Lift Modeling

▸ Uplift Modeling

Network Analysis

▸ Link Mining and Link Discovery

Network Clustering

▸ Graph Clustering

Networks with Kernel Functions

▸ Radial Basis Function Networks

Neural Networks

Neural networks are learning algorithms based on a loose analogy of how the human brain functions. Learning is achieved by adjusting the weights on the connections between nodes, which are analogous to synapses and neurons.

Cross-References

▶ Radial Basis Function Networks

Neuro-Dynamic Programming

▶ Value Function Approximation

Neuroevolution

Risto Miikkulainen
Department of Computer Science, The
University of Texas at Austin, Austin, TX, USA

Abstract

Neuroevolution is a method for modifying neural network weights, topologies, or ensembles in order to learn a specific task. Evolutionary computation is used to search for network parameters that maximize a fitness function that measures performance in the task. Compared to other neural network learning methods, neuroevolution is highly general, allowing learning without explicit targets, with nondifferentiable activation functions, and with recurrent networks. It can also be combined with standard neural network learning to, e.g., model biological adaptation. Neuroevolution can also be seen as a policy search method for reinforcement learning problems, where it is well suited to continuous domains and to domains where the state is only partially observable.

Synonyms

Evolving neural networks; Genetic neural networks

Motivation and Background

The primary motivation for neuroevolution is to be able to train neural networks in sequential decision tasks with sparse reinforcement information. Most neural network learning is concerned with supervised tasks, where the desired behavior is described in terms of a corpus of input-output examples. However, many learning tasks in the real world do not lend themselves to the supervised learning approach. For example, in game playing, vehicle control, and robotics, the optimal actions at each point in time are not always known; only after performing several actions it is possible to get information about how well they worked, such as winning or losing the game. Neuroevolution makes it possible to find a neural network that optimizes such behavior given only sparse information about how well the networks are doing, without direct information about what exactly they should be doing.

The main benefit of neuroevolution compared to other reinforcement learning (RL) methods is that it allows representing continuous state and action spaces and disambiguating hidden states naturally. Network activations are continuous, and the network generalizes well between continuous values, largely avoiding the state explosion problem that plagues many reinforcement learning approaches. Recurrent networks can encode memories of past states and actions, making it possible to learn in partially observable Markov decision process (POMDP) environments that are difficult for many RL approaches.

Compared to other neural network learning methods, neuroevolution is highly general. As long as the performance of the networks can be evaluated over time and the behavior of the network can be modified through evolution, it can be applied to a wide range of network architectures, including those with nondifferentiable activation functions and recurrent and higher-order connections. While most neural learning algorithms focus on modifying the weights only, neuroevolution can be used to optimize other aspects of the networks as well, including activation functions and network topologies.

Neuroevolution allows combining evolution over a population of solutions with lifetime learning in individual solutions: the evolved networks can each learn further through, e.g., backpropagation or Hebbian learning. The approach is therefore well suited to understanding biological adaptation and for building artificial life systems.

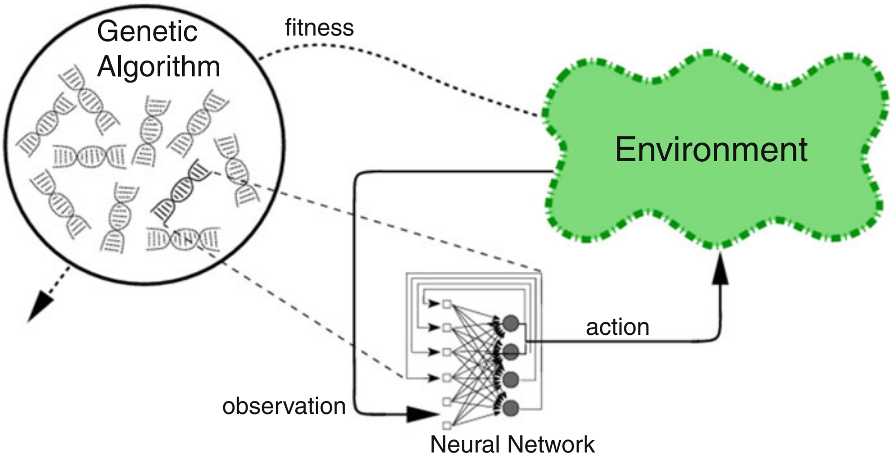
Structure of the Learning System

Basic Methods

In neuroevolution, a population of genetic encodings of neural networks is evolved in order to find a network that solves the given task. Most neuroevolution methods follow the usual generate-and-test loop of evolutionary algorithms (Fig. 1). Each encoding in the population (a genotype) is chosen in turn and decoded into the corresponding neural network (a phenotype). This network is then employed in the task and its performance over time measured, obtaining a fitness value for the corresponding genotype. After all members of the population have been evaluated in this manner, genetic operators are used to create the next generation of the population. Those encodings with the highest fitness are mutated and crossed over with each other, and the resulting offspring replaces the genotypes with the lowest fitness in the population. The process therefore constitutes an intelligent parallel search toward better genotypes and continues until a network with a sufficiently high fitness is found.

Several methods exist for evolving neural networks depending on how the networks are encoded. The most straightforward encoding, sometimes called conventional neuroevolution (CNE), is formed by concatenating the numerical values for the network weights (either binary or floating point) (Floreano et al. 2008; Yao 1999; Schaffer et al. 1992). This encoding allows evolution to optimize the weights of a fixed neural network architecture, an approach that is easy to implement and is practical in many domains.

In more challenging domains, the CNE approach suffers from three problems: The method may cause the population to converge before a solution is found, making further progress difficult (i.e., premature convergence); similar networks, such as those where the order of nodes is different, may have different encodings, and much effort is wasted in trying to optimize them in parallel (i.e., competing conventions); a large number of parameters need



Neuroevolution, Fig. 1 **Evolving Neural Networks.** A population of genetic neural network encodings (genotypes) is first created. At each iteration of evolution (generation), each genotype is decoded into a neural network (phenotype), which is evaluated in the task, resulting in a fitness value for the genotype. Crossover and mutation among the genotypes with the highest fitness are then used to generate the next generation to be optimized at once, which is difficult through evolution.

More sophisticated encodings have been devised to alleviate these problems. One approach is to run the evolution at the level of solution components instead of full solutions. That is, instead of a population of complete neural networks, a population of network fragments, neurons, or connection weights is evolved (Moriarty et al. 1999; Gomez et al. 2008; Potter and Jong 2000). Each individual is evaluated as part of a full network, and its fitness reflects how well it cooperates with other individuals in forming a full network. Specifications for how to combine the components into a full network can be evolved separately, or the combination can be based on designated roles for subpopulations. In this manner, the complex problem of finding a solution network is broken into several smaller subproblems; evolution is forced to maintain diverse solutions, and competing conventions and the number of parameters is drastically reduced.

Another approach is to evolve the network topology, in addition to the weights. The idea is that topology can have a large effect on function and evolving appropriate topologies can achieve good performance faster than evolving weights only (Angeline et al. 1994; Floreano et al. 2008; Yao 1999; Stanley and Miikkulainen 2004). Since topologies are explicitly specified, competing conventions are largely avoided. It is also possible to start evolution with simple solutions and gradually make them more complex, a process that takes place in biology and is a powerful approach in machine learning in general. Speciation according to the topology can be used to avoid premature convergence and to protect novel topological solutions until their weights have been sufficiently optimized.

All of the above methods map the genetic encoding directly to the corresponding neural network, i.e., each part of the encoding corresponds to a part of the network and vice versa. Indirect encoding, in contrast, specifies a process through which the network is constructed, such as cell division or generation through a grammar or through patterns generated by another neural network (Floreano et al. 2008; Yao 1999; Stanley and Miikkulainen 2003; Gruau and Whitley 1993; Stanley et al. 2009). Such an encoding can be highly compact and also take advantage of modular solutions. The same structures can be repeated with minor modifications, as they often are in biology. It is, however, difficult to optimize solutions produced by indirect encoding, and realizing its full potential is still future work.

Another approach is to evolve an ensemble of neural networks to solve the task together, instead of a single network (Liu et al. 2000). This approach takes advantage of the diversity in the population: Different networks learn different parts or aspects of the training data, and together the whole ensemble can perform better than a single network. Diversity can be created through speciation and negative correlation, encouraging useful specializations to emerge. The approach can be used to design ensembles for classification problems, but it can also be extended to control tasks.

Extensions

The basic mechanisms of neuroevolution can be augmented in several ways, making the process more efficient and extending it to various applications. One of the most basic ones is incremental evolution or shaping: Evolution is began on a simple task, and once that is mastered, the solutions are evolved further on a more challenging task and, through a series of such transfer steps, eventually on the actual goal task itself (Gomez et al. 2008). Shaping can be done by changing the environment, such as increasing the speed of the opponents, or by changing the fitness function, e.g., by rewarding gradually more complex behaviors. It is often possible to solve challenging tasks by approaching them incrementally even when they cannot be solved directly.

Many extensions to evolutionary computation methods apply particularly well to neuroevolution. First, intelligent mutation techniques such as those employed in evolutionary strategies are effective because the weights often have suitable correlations (Igel 2003). Second, networks can be evolved through coevolution (Stanley and Miikkulainen 2004; Chellapilla and Fogel 1999). A coevolutionary arms race can be established, e.g., based on complexification of network topology: As the network becomes gradually more complex, evolution is likely to elaborate on existing behaviors instead of replacing them. Third, behavioral diversity and novelty can be defined naturally in terms of network behavior, leading to methods that discover novel solutions (Lehman and Stanley 2010; Mouret and Doncieux 2012).

On the other hand, several extensions utilize the special properties of the neural network phenotype. For instance, neuron activation functions, initial states, and learning rules can be evolved to fit the task (Floreano et al. 2008; Yao 1999; Schaffer et al. 1992). It is possible to evolve modular network architectures, e.g., as a separate mutation or through minimizing wiring length, and thus discover how complex behavior arises from a combination of low-level behaviors (Clune et al. 2013; Schrum 2014). Most significantly, evolution can be combined with other neural network learning methods (Floreano et al. 2008). In such approaches, evolution usually provides the initial network, which then adapts further during its evaluation in the task. The adaptation can take place through Hebbian learning, thereby strengthening those existing behaviors that are invoked often during evaluation. Alternatively, supervised learning such as backpropagation can be used, provided targets are available. Even if the optimal behaviors are not known, such training can be useful: Networks can be trained to imitate the most successful individuals in the population, or part of the network can be trained in a related task such as predicting the next inputs or evaluating the utility of actions based on values obtained through Q-learning. The weight changes may be encoded back into the genotype, implementing Lamarckian evolution; alternatively, they may affect selection through the Baldwin effect, i.e., networks that learn well will be selected for reproduction even if the weight changes themselves are not inherited (Ackley and Littman 1992; Gruau and Whitley 1993; Bryant and Miikkulainen 2007).

There are also several ways to bias and direct the learning system using human knowledge. For instance, human-coded rules can be encoded in partial network structures and incorporated into the evolving networks as structural mutations. Such knowledge can be used to implement initial behaviors in the population, or it can serve as advice during evolution (Miikkulainen et al. 2006). In cases where rule-based knowledge is not available, it may still be possible to obtain examples of human behavior. Such examples can then be incorporated into evolution, either as components of fitness or by explicitly training the evolved solutions toward human behavior through, e.g., backpropagation (Bryant and Miikkulainen 2007). Similarly, knowledge about the task and its components can be utilized in designing effective shaping strategies. In this manner, human expertise can be used to bootstrap and guide evolution in difficult tasks, as well as direct it toward the desired kinds of solutions.

Applications

Neuroevolution methods are powerful especially in continuous domains of reinforcement learning and those that have partially observable states. For instance, in the benchmark task of balancing the inverted pendulum without velocity information (making the problem partially observable), the advanced methods have been shown to find solutions two orders of magnitude faster than value function-based reinforcement learning methods (measured by number of evaluations, Gomez et al. 2008). They can also solve harder versions of the problem, such as balancing two poles simultaneously.

The method is powerful enough to make many real-world applications of reinforcement learning possible. The most obvious area is adaptive, nonlinear control of physical devices. For instance, neural network controllers have been evolved to drive mobile robots, automobiles, and even rockets (Valsalam et al. 2013; Togelius and Lucas 2006; Gomez and Miikkulainen 2003; Nolfi and Floreano 2000; Bongard 2011). The control approach have been extended to optimize systems such as chemical processes, manufacturing systems, and computer systems. A crucial limitation with current approaches is that the controllers usually need to be developed in simulation and transferred to the real system. Evolution is strongest as an off-line learning method where it is free to explore potential solutions in parallel.

Evolution of neural networks is a natural tool for problems in artificial life. Because networks implement behaviors, it is possible to design neuroevolution experiments on how behaviors such as foraging, pursuit and evasion, hunting and herding, collaboration, and even communication may emerge in response to environmental pressure (Werner and Dyer 1992; Nolfi and Floreano 2000). It is possible to evolve the morphology and control together to create agents with natural movement (Lessin et al. 2013; Bongard 2011) and to analyze the evolved circuits and understand how they map to function, leading to insights into biological networks (Keinan et al. 2006). The evolutionary behavior approach is also useful for constructing characters in artificial environments, such as games and simulators. Non-player characters in current video games are usually scripted and limited; neuroevolution can be used to evolve complex behaviors for them and even adapt them in real time (Miikkulainen et al. 2006; Risi and Togelius 2014).

Programs and Data

Software for the NEAT method for evolving network weights and topologies, and for the ESP and CoSyNE methods for evolving neurons and weights to form networks, is available at nn.cs.utexas.edu/?neuroevolution. Software for HyperNEAT indirect neuroevolution method is available at eplex.cs.ucf.edu/hyperNEATpage.

PyBrain (pybrain.org) and Sferes2 (github.com/jbmouret/sferes2) are general machine learning and evolutionary computation packages that include neuroevolution methods.

The OpenNERO software for evolving intelligent multiagent behavior in simulated environments is at http://opennero.googlecode.com.

Cross-References

▸ Evolutionary Computation
▸ Reinforcement Learning

Recommended Reading

Ackley D, Littman M (1992) Interactions between learning and evolution. In: Langton CG, Taylor C, Farmer JD, Rasmussen S (eds) Artificial life II. Addison-Wesley, Reading, pp 487–509

Angeline PJ, Saunders GM, Pollack JB (1994) An evolutionary algorithm that constructs recurrent neural networks. IEEE Trans Neural Netw 5:54–65

Bongard J (2011) Morphological change in machines accelerates the evolution of robust behavior. Proc Natl Acad Sci USA 108:1234–1239

Bryant BD, Miikkulainen R (2007) Acquiring visibly intelligent behavior with example-guided neuroevolution. In: Proceedings of the twenty-second national conference on artificial intelligence. AAAI, Menlo Park

Chellapilla K, Fogel DB (1999) Evolution, neural networks, games, and intelligence. Proc IEEE 87:1471–1496

Clune J, Mouret J-B, Lipson H (2013) The evolutionary origins of modularity. Proc R Soc B Biol Sci 280(1755):20122863

Floreano D, Dürr P, Mattiussi C (2008) Neuroevolution: from architectures to learning. Evol Intell 1:47–62

Gomez F, Miikkulainen R (2003) Active guidance for a finless rocket using neuroevolution. In: Proceedings of the genetic and evolutionary computation conference. Morgan Kaufmann, San Francisco, pp 2084–2095

Gomez F, Schmidhuber J, Miikkulainen R (2008) Accelerated neural evolution through cooperatively coevolved synapses. J Mach Learn Res 9:937–965

Gruau F, Whitley D (1993) Adding learning to the cellular development of neural networks: evolution and the Baldwin effect. Evol Comput 1:213–233

Igel C (2003) Neuroevolution for reinforcement learning using evolution strategies. In: Sarker R, Reynolds R, Abbass H, Tan KC, McKay B, Essam D, Gedeon T (eds) Proceedings of the 2003 congress on evolutionary computation. IEEE, Piscataway, pp 2588–2595

Keinan A, Sandbank B, Hilgetag CC, Meilijson I, Ruppin E (2006) Axiomatic scalable neurocontroller analysis via the Shapley value. Artif Life 12:333–352

Lehman J, Stanley KO (2010) Abandoning objectives: evolution through the search for novelty alone. Evol Comput 2011:189–223

Lessin D, Fussell D, Miikkulainen R (2013) Open-ended behavioral complexity for evolved virtual

creatures. In: Proceedings of the genetic and evolutionary computation conference, Amsterdam
Liu Y, Yao X, Higuchi T (2000) Evolutionary ensembles with negative correlation learning. IEEE Trans Evol Comput 4:380–387
Miikkulainen R, Bryant BD, Cornelius R, Karpov IV, Stanley KO, Yong CH (2006) Computational intelligence in games. In: Yen GY, Fogel DB (eds) Computational intelligence: principles and practice, Piscataway. IEEE Computational Intelligence Society
Moriarty DE, Schultz AC, Grefenstette JJ (1999) Evolutionary algorithms for reinforcement learning. J Artif Intell Res 11:199–229
Mouret J-B, Doncieux S (2012) Encouraging behavioral diversity in evolutionary robotics: an empirical study. Evol Comput 20:91–133
Nolfi S, Floreano D (2000) Evolutionary robotics. MIT, Cambridge
Potter MA, Jong KAD (2000) Cooperative coevolution: an architecture for evolving coadapted subcomponents. Evol Comput 8:1–29
Risi S, Togelius J (2014) Neuroevolution in games: state of the art and open challenges ArXiv e-prints. E-print no. 1410.7326
Schaffer JD, Whitley D, Eshelman LJ (1992) Combinations of genetic algorithms and neural networks: a survey of the state of the art. In: Whitley D, Schaffer J (eds) Proceedings of the international workshop on combinations of genetic algorithms and neural networks. IEEE Computer Society Press, Los Alamitos, pp 1–37
Schrum J (2014) Evolving multimodal behavior through modular multiobjective neuroevolution. Ph.D. thesis, The University of Texas at Austin, Austin. Technical report TR-14-07
Stanley KO, D'Ambrosio DB, Gauci J (2009) A hypercube-based encoding for evolving large-scale neural networks. Artif Life 15(2):185–212
Stanley KO, Miikkulainen R (2003) A taxonomy for artificial embryogeny. Artif Life 9(2):93–130
Stanley KO, Miikkulainen R (2004) Competitive coevolution through evolutionary complexification. J Artif Intell Res 21:63–100
Togelius J, Lucas SM (2006) Evolving robust and specialized car racing skills. In: IEEE congress on evolutionary computation. IEEE, Piscataway, pp 1187–1194
Valsalam V, Hiller J, MacCurdy R, Lipson H, Miikkulainen R (2013) Constructing controllers for physical multilegged robots using the enso neuroevolution approach. Evol Intell 14:303–331
Werner GM, Dyer MG (1992) Evolution of communication in artificial organisms. In: Langton CG, Taylor C, Farmer JD, Rasmussen S (eds) Proceedings of the workshop on artificial life (ALIFE '90). Addison-Wesley, Reading, pp 659 –687
Yao X (1999) Evolving artificial neural networks. Proc IEEE 87(9):1423–1447

Neuron

Risto Miikkulainen
Department of Computer Science, The University of Texas at Austin, Austin, TX, USA

Synonyms

Node; Unit

Definition

Neurons carry out the computational operations of a network; together with connections (see ▶ Topology of a Neural Network, ▶ Weight), they constitute the neural network. Computational neurons are highly abstracted from their biological counterparts. In most cases, the neuron forms a weighted sum of a large number of inputs (activations of other neurons), applies a nonlinear transfer function to that sum, and broadcasts the resulting output activation to a large number of other neurons. Such activation models the firing rate of the biological neuron, and the nonlinearity is used to limit it to a certain range (e.g., 0 or 1 with a threshold, $(0, 1)$ with a sigmoid, $(-1, 1)$ with a hyperbolic tangent, or $(0, \infty)$ with an exponential function). Each neuron may also have a bias weight, i.e., a weight from a virtual neuron that is always maximally activated, which the learning algorithm can use to adjust the input sum quickly into the most effective range of the nonlinearity. Alternatively to firing rate neurons, the firing events (i.e., spikes or action potentials) of the neuron can be represented explicitly. In such an integrate-and-fire approach, each spike causes a change in the neuron's membrane potential that decays over time; an output spike is generated if the potential exceeds a threshold (see ▶ Biological Learning: Synaptic Plasticity, Hebb Rule and Spike Timing Dependent Plasticity). In contrast, networks such as ▶ self-organizing maps and ▶ radial basis function networks abstract the firing rate further into a measure of similarity (or distance) between the neuron's input weight vector and the vector of input activities. Learning in neural networks usually takes place by adjusting the weights on the input connections of the neuron, and can also include adjusting the parameters of the nonlinear transfer function, or the neuron's connectivity with other neurons. In this manner, the neuron converges information from other neurons, makes a simple decision based on it, broadcasts the result widely, and adapts.

Node

▶ Neuron

No-Free-Lunch Theorem

A theorem establishing that performance on test data cannot be deduced from performance on training data. It follows that the justification for any particular learning algorithm must be based on an assumption that nature is uniform in some way. Since different machine learning algorithms make such different assumptions, no-free-lunch theorems have been used to argue that it not possible to deduce that any algorithm is superior to any other from first principles. Thus "good" algorithms are those whose ▶ inductive bias matches the way the world happens to be.

Further Reading

Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82

Nogood Learning

Nogood learning is a ▶ deductive learning technique used for the purpose of ▶ intelligent backtrackings in constraint satisfaction. The approach analyzes failures at backtracking points and derives sets of variable bindings, or *nogoods*, that will never lead to a solution. These nogood constraints can then be used to prune later search nodes.

Noise

The training data for a learning algorithm is said to be *noisy* if the data contain errors. Errors can be of two types:

- A *measurement error* occurs when some attribute values are incorrect or inaccurate. Note that measurement of physical properties by continuous values is always subject to some error.
- In supervised learning, *classification error* means that a training example has an incorrect class label.

In addition to errors, training examples may have ▶ missing attribute values. That is, the values of some attribute values are not recorded.

Noisy data can cause learning algorithms to fail to converge to a concept description or to build a concept description that has poor classification accuracy on unseen examples. This is often due to ▶ over fitting.

For methods to minimize the effects of noise, see ▶ Overfitting.

Nominal Attribute

A **nominal attribute** assumes values that classify data into mutually exclusive (nonoverlapping), exhaustive, unordered categories. See ▶ Attribute and ▶ Measurement Scales.

Nonparametric Bayesian

▶ Gaussian Process

Nonparametric Cluster Analysis

▶ Density-Based Clustering

Non-Parametric Methods

▶ Instance-Based Learning

Nonstandard Criteria in Evolutionary Learning

Michele Sebag
CNRS – INRIA – Université Paris-Sud, Orsay,
France

Introduction

Machine learning (ML), primarily concerned with extracting models or hypotheses from data, comes into three main flavors: ▶ supervised learning also known as ▶ classification or ▶ regression (Bishop 2006; Duda et al. 2001; Han and Kamber 2000), ▶ unsupervised learning also known as ▶ clustering (Ben-David et al. 2005), and ▶ reinforcement learning (Sutton and Barto 1998).

All three types of problems can be viewed as optimization problems. The ML core task is to define a *learning criterion* (i.e., the function to be optimized) such that it enforces (i) the statistical relevance of the solution; (ii) the well-posedness of the underlying optimization problem. Since evolutionary computation (see ▶ Evolutionary Algorithms) makes it possible to handle ill-posed optimization problems, the field of evolutionary learning (Holland 1986) has investigated quite a few nonstandard learning criteria and search spaces. Only supervised ML will be considered in the following. Unsupervised learning has hardly been touched upon in the evolutionary computation (EC) literature; regarding reinforcement learning, the interested reader is referred to the entries related to ▶ evolutionary robotics and control.

The entry will first briefly summarize the formal background of supervised ML and its two mainstream approaches for the last decade, namely support vector machines (SVMs) (Cristianini and Shawe-Taylor 2000; Schölkopf et al. 1998; Vapnik 1995) and ensemble learning (Breiman 1998; Dietterich 2000; Schapire 1990). Thereafter and without pretending to exhaustivity, this entry will illustrate some innovative variants of these approaches in the literature, building upon the evolutionary freedom of setting and tackling optimization problems.

Nonstandard Criteria in Evolutionary Learning, Table 1 Excerpt of a dataset in a failure identification problem (binary classification). Instance space X is the cross product of all attribute domains: for example, attribute *Temperature* ranges in R, attribute *Material* ranges in $\{Ni, Fe, \ldots\}$. *Label space* Y *is binary*

	Temperature	Material	Aging	Label
$\mathbf{x}_1$	118.2	Ni	No	Failure
$\mathbf{x}_2$	76.453	Fe	Yes	OK

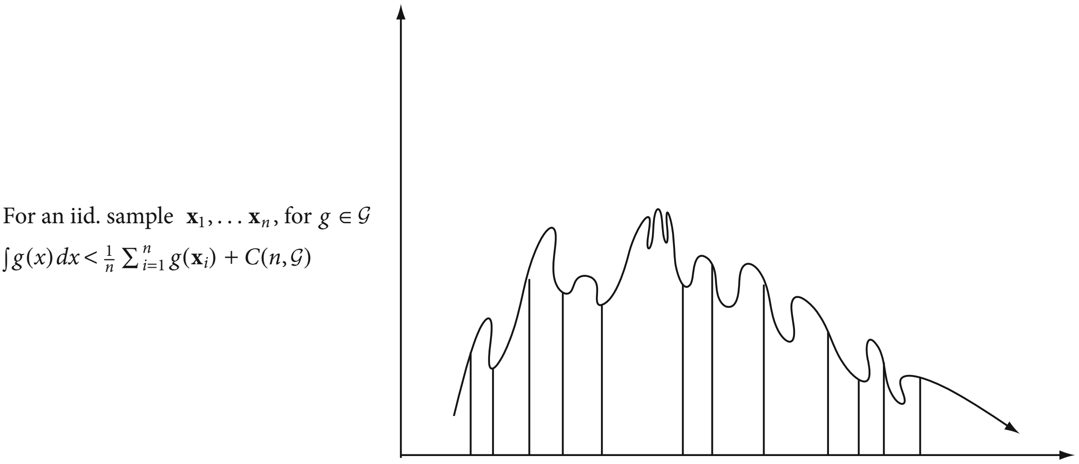
Formal Background

Supervised learning exploits a dataset $\mathcal{E} = \{(\mathbf{x}_i, y_i), \mathbf{x}_i \in X, y_i, \in Y, i = 1 \ldots n\}$, where X stands for the instance space (e.g., $\mathbb{R}^d$), Y is the label space, and $(\mathbf{x}_i, y_i)$ is a labeled example, as depicted in Table 1. Supervised learning is referred to as *classification* (respectively *regression*) when Y is a finite set (respectively when $Y = \mathbb{R}$).

The ML goal is to find a hypothesis or classifier $h : X \mapsto Y$ such that $h(x)$ is "sufficiently close" to the true label y of x for any x ranging in the instance domain. It is generally assumed that the available examples are independently and identically distributed (iid) after a probability distribution P_{XY} on $X \times Y$. Letting $\ell(y', y)$ denote the loss incurred by labeling $\mathbf{x}$ as y' instead of its true label y, the learning criterion is most naturally defined as the expectation of the loss, or *generalization error*, to be minimized, where $\mathcal{H}$ denotes the hypothesis space:

$$\text{Find } h^* = arg\ \min\{\mathcal{F}(h) = \int \ell(h(x), y) dP(x, y), h \in \mathcal{H}\}$$

The generalization error however is not computable, since the joint distribution P_{XY} of in-


For an iid. sample $\mathbf{x}_1, \ldots \mathbf{x}_n$, for $g \in \mathcal{G}$
$\int g(x)\,dx < \frac{1}{n} \sum_{i=1}^{n} g(\mathbf{x}_i) + C(n, \mathcal{G})$

Nonstandard Criteria in Evolutionary Learning, Fig. 1 Bounding the integral from the empirical average depending on the uniform sample size and the class of functions $\mathcal{G}$ at hand

stances and labels is unknown; only its approximation on the training set, referred to as *empirical error*, can be computed as follows:

$$\mathcal{F}_e(h) = \frac{1}{n} \sum_{i=1}^{n} \ell(h(\mathbf{x}_i), y_i)$$

Using results from the theory of measure and integration, the generalization error is upper bounded by the empirical error, plus a term reflecting the number of examples and the regularity of the hypothesis class (Fig. 1).

Note that minimizing the empirical error alone leads to the infamous *overfitting* problem: while the predictive accuracy on the training set is excellent, the error on a (disjoint) test set is much higher. All learning criteria thus involve a trade-off between the empirical error and a so-called regularization term, providing good guarantees (upper bound) on the generalization error.

In practice, learning algorithms also involve hyper-parameters (e.g., the weight of the regularization term). These are adjusted using cross-validation using a grid search (EC approaches have also been used to find optimal learning hyperparameters, ranging from the topology of neural nets (Miikkulainen et al. 2003), to the kernel parameters in SVM (Friedrichs and Igel 2005; Mierswa 2006) The dataset is divided into K subsets with same class distribution; hypothesis h_i is learned from the training set made of all subsets except the i-th and the empirical error of h_i is measured on the ith subset. An approximation of the generalization error is provided by the average of the h_i errors when $i = 1 \ldots K$, referred to as cross-fold error, and the hyperparameter setting is empirically determined to minimize the cross-fold error.

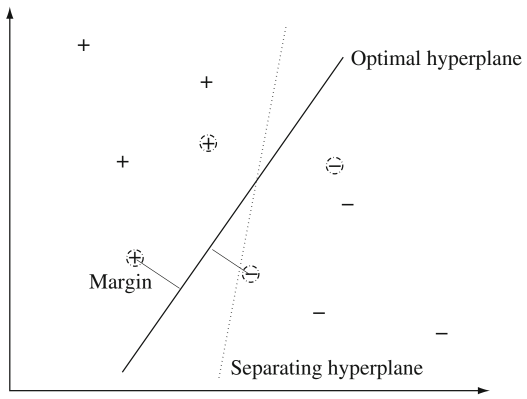
Support Vector Machines

Considering a real-valued instance space $(X = \mathbb{R}^D)$, a linear ▸ support vector machine (SVM) (Boser et al. 1992) constructs the separating hyperplane (where $< a, b >$ stands for the dot product of vectors a and b):

$$h(\mathbf{x}) = < w, \mathbf{x} > +b$$

which maximizes the margin that is, the minimal distance between the examples and the hyperplane, when such separating hyperplanes exists (Fig. 2). A slightly more complex formulation, involving the so-called slack variables $\mathbf{x}i_i$, is defined to deal with noise (Cortes and Vapnik 1995).

The function to be optimized, the L_2 norm of the hyperplane normal vector w, is quadratic; using Lagrange multipliers to account for the constraints gives rise to the so-called dual formulation. Let us call *support vectors* those examples



without noise

$$\text{Minimize } \tfrac{1}{2}||w||^2$$

s.t. for $i = 1$ to n

$$y_i(< w, \mathbf{x}_i > +b) \geq 1$$

with noise

$$\text{Minimize } \tfrac{1}{2}||w||^2 + C\textstyle\sum_{i=1}^{n} \mathbf{x}i_i$$

s.t. for $i = 1$ to n

$$y_i(< w, \mathbf{x}_i > +b) \geq 1 - \mathbf{x}i_i; \quad \mathbf{x}i_i \geq 0$$

Nonstandard Criteria in Evolutionary Learning, Fig. 2 Linear support vector machines. The optimal hyperplane is the one maximizing the minimal distance to the examples

for which the constraint is active (Lagrange multiplier $\alpha_i > 0$), then it becomes

$$h(\mathbf{x}) = \sum y_i \alpha_i < \mathbf{x}_i, \mathbf{x} > +b \quad \text{with } \alpha_i > 0;$$
$$\sum \alpha_i y_i = 0$$

As will be seen in section "Evolutionary Regularization" this formulation defines a search space, which can be directly explored by EC (Mierswa 2007).

Obviously however, linear classifiers are limited. The power of SVMs comes from the so-called kernel *trick*, naturally exporting the SVM approach to nonlinear hypothesis spaces. Let us map the instance space X onto some *feature space* X' via mapping Φ. If the scalar product on X' can be computed in X (e.g., $< \Phi(\mathbf{x}), \Phi(\mathbf{x}') >=_{\text{def}} K(x, x')$) then a linear classifier in X' (nonlinear with reference to X) is given as $h(\mathbf{x}) = \sum_i y_i \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b$. The only requirement is to use a positive definite kernel (ensuring that the underlying optimization problem is well posed). Again, this requirement can be relaxed in the evolutionary learning framework (Mierswa 2006).

Among the most widely used kernels are the Gaussian kernel $\left(K(x, x') = \exp\left\{-\frac{||x-x'||^2}{\sigma^2}\right\}\right)$ and the polynomial kernel $(K(x, x') = (< x, x' > +c)^d)$. The kernel parameters σ, c, d, referred to as learning hyper-parameters, have been tuned by some authors using EC, as well as the kernel itself (see among others Friedrichs and Igel 2005; Gagné et al. 2006; Mierswa 2006).

Ensemble Methods

The other mainstream approach in supervised learning, ▸ ensemble learning (EL), relies on somewhat different principles. Schapire's seminal paper, *The strength of weak learnability*, exploring the relationship between *weak learnability* (ability of building a hypothesis slightly better than random guessing, whatever the distribution of the dataset is (C)) and *strong learnability* (ability of building a hypothesis with arbitrarily high predictive accuracy), established a major and counterintuitive result: strong and weak learnability are equivalent (Schapire 1990). The idea behind the proof is that combining many weak hypotheses learned under different distributions yields an arbitrarily accurate hypothesis. As the errors of the weak hypotheses should not concentrate in any particular region of the instance space (for condition C to hold), the law of large numbers states that averaging them leads to exponentially decrease the empirical error.

Two main EL approaches have been investigated in the literature. The first one, ▸ bagging (Breiman 1998), builds a large number of independent hypotheses; the source of variations is bootstrapping (uniformly selecting the training set with replacement from the initial dataset); or varying the parameters of the learning algorithm; or subsampling the features considered at each step of the learning process (Amit et al. 1997; Breiman 2001). The final classifier is usually obtained by averaging these solutions.

The other EL approach, ▸ boosting (Freund and Shapire 1996), iteratively builds a sequence of hypotheses, where each h_i somehow is in charge of correcting the mistakes of $h_1, \ldots h_{i-1}$. Specifically, a distribution $\mathcal{W}_t$ is defined on the training set at step t, with $\mathcal{W}_0$ being the uniform distribution. At step t, the weight of every example misclassified by ht is increased (multiplied by $\exp\{-h_t(\mathbf{x}_i).\ h_i\}$; then a normalization step follows to ensure that $\mathcal{W}_{t+1}$ still sums to 1); hypothesis h_{t+1} will thus focus on the examples misclassified by h_t. Finally, the classifier is defined as the weighted vote of all h_t.

The intuition behind boosting is that not all examples are equal: some examples are more difficult than others (more hypotheses misclassify them) and the learning process should thus focus on these examples (with the caveat that a difficult example might be so because it is noisy). Interestingly, the intuition that examples are not equal has been formalized in terms of coevolution (When designing a program, the fitness of the candidate solutions is computed after some test cases; for the sake of accuracy and feasability, the difficulty and number of test cases must be commensurate with the competence of the current candidate solutions. Hillis defined a competitive coevolution setting between the program species and the test case species: while programs aim at solving test cases, test cases aim at defeating candidate programs. This major line of research however is outside the scope of evolutionary learning as it assumes that the whole distribution PXY is known.) by D. Hillis in the early 1990s (Hillis 1990).

Many empirical studies suggest that boosting is more effective than bagging (with some caveat in the case of noisy domains), thanks to the higher diversity of the boosting ensemble (Dietterich 2000; Margineantu and Dietterich 1997).

In the ensemble learning framework, the *margin* of an example $\mathbf{x}$ is defined as the difference between the (cumulated weight or number) of hypotheses labeling $\mathbf{x}$ as positive, and those labeling $\mathbf{x}$ as negative. Like in the SVM framework, the margin of an example reflects the confidence of its classification (how much this example should be perturbed for its label to be modified).

Learning Criteria

Learning criterion and *fitness function* will be used interchangeably in the following. Since Holland's seminal papers on evolutionary learning (Holland 1975, 1986), the most used learning criterion is the predictive accuracy on the available dataset. After the early 1990s however, drawbacks related to either learning or evolutionary issues motivated the design of new fitness functions.

Evolutionary Regularization

In the ▸ genetic programming field, the early use of more sophisticated learning criteria was motivated by the so-called bloat phenomenon (Banzhaf and Langdon 2002; Poli 2008), that is, the uncontrolled growth of the solution size as evolution goes on. Two main approaches have been considered. The first one boils down to regularization (section "Formal Background"): the fitness function is composed of the predictive accuracy plus an additional term meant to penalize large-sized solutions (Blickle 1996). The tricky issue of course is how to adjust the weight of the penalization term; the statistical ML theory offers no principled solution to this issue (except in an asymptotic perspective, when the number of training examples goes to infinity (Gelly et al. 2006)); thus, the weight is adjusted empirically using cross-validation (section "Formal Background").

Another approach (Blickle 1996) is based on the use of two fitness functions during the same evolution run, after the so-called behavioral memory paradigm (Schoenauer and Xanthakis 1993). In a first phase, the population is evolved to maximize the predictive accuracy. In a second phase, the optimization goal becomes to minimize the solution size *while preserving the predictive accuracy* reached in the former phase. As could have been expected, this second approach also depends upon the careful empirical adjustment of hyper-parameters (when to switch from one phase to another one).

Another approach is to consider regularized learning as a multi-objective optimization problem, avoiding the computationally heavy tuning of the regularization weight (Note however that in the case where the regularization involves the L_1 norm of the solution, the Pareto front can be analytically derived using the celebrated LASSO algorithm (Hastie et al. 2004; Tibshirani 1996).). Mierswa (2007) applies multi-objective evolutionary optimization, specifically NSGA-II ((Deb et al. 2000); see the Multi-Objective Evolutionary Optimization entry in this encyclopedia), to the simultaneous optimization of the margin and the error. The search space is nicely and elegantly derived from the dual form of SVMs (section "Support Vector Machines"): it consists of vectors $(\alpha_1, \ldots \alpha_n)$, where most α_i are zero and $\sum_i \alpha_i y_i = 0$. A customized mutation operator, similar in spirit to the sequential minimization optimization proposed by Platt (1999), enables to explore the solutions with few support vectors. The Pareto front shows the trade-off between the regularization term and the training error. At some point however, a hold-out (test set) needs be used to detect and avoid overfitting solutions, boiling down to cross-validation. Another multi-objective optimization learning is proposed by Suttorp and Igel (2006) (see section "AUC: Area Under the ROC Curve").

Ensemble Learning and Boosting

Ensemble learning and evolutionary computation share two main original features. Firstly, both rely on a population of candidate solutions; secondly, the diversity of these solutions commands the effectiveness of the approach. It is no surprise therefore that evolutionary ensemble learning, tightly coupling EC and EL, has been intensively investigated in the last decade (Another exploitation of the hypotheses built along independent evolutionary learning runs concerns feature selection (Jong et al. 2004), which is outside the scope of this entry.)

A family of diversity-oriented learning criteria has been investigated by Xin Yao and collaborators, switching the optimization goal from "learning the best hypothesis" toward "learning the best ensemble" Monirul Islam and Yao (2008). The hypothesis space is that of neural networks (NNs). Nonparametric and parametric operators are used to simultaneously optimize the neural topology and the NN weights. Among parametric operators is the gradient-based backpropagation (BP) algorithm to locally optimize the weights (Rumelhart and McClelland 1990), combined with simulated annealing to escape BP local minima.

Liu et al. (2000) enforce the diversity of the networks using a *negative correlation* learning criterion. Specifically, the BP algorithm is modified by replacing the error of the t-th NN on the i-th example with a weighted sum of this error and the error of the ensemble of the other NNs; denoting H_{-t} the ensemble made of all NNs but the tth one:

$$(h_t(x_i) - y_i)^2 \rightarrow (1-\lambda)(h_t(x_i) - y_i)^2 + \lambda(H_{-t}(x_i) - y_i)^2$$

Moreover, ensemble negative correlation–based learning exploits the fact that not all examples are equal, along the same line as boosting (section "Ensemble Methods"): to each training example is attached a weight, reflecting the number of hypotheses that misclassify it; finally the fitness associated to each network is the sum of the weights of all examples it correctly classifies. While this approach nicely suggests that ensemble learning is a multiple objective optimization (MOO) problem (minimize the error rate and maximize the diversity), it classically handles the MOO problem as a fixed weighted sum of the objectives (the value of parameter λ is fixed by the user).

The MOO perspective is further investigated by Chandra and Yao in the DIVACE system, enforcing the multilevel evolution of ensemble of classifiers (Chandra and Yao 2006a,b). In (Chandra and Yao 2006b), the top-level evolution simultaneously minimizes the error rate (accuracy) and maximizes the negative correlation (diversity). In (Chandra and Yao 2006a), the negative correlation-inspired criterion is replaced by a *pairwise failure crediting*; the difference concerns the misclassification of examples that are correctly classified by other classifiers. Several heuristics have been investigated to construct the ensemble from the last population, based on averaging the hypothesis values, using the (weighted) vote of all hypotheses, or selecting a subset of hypotheses, for example, by clustering the final hypothesis population after their phenotypic distance, and selecting a hypothesis in each cluster.

Gagné et al. (2007) tackle both the construction of a portfolio of classifiers, and the selection of a subset thereof, either from the final population only as in (Chandra and Yao 2006a,b), or from all generations. In order to do so, a reference set of classifiers is used to define a dynamic optimization problem: the fitness of a candidate hypothesis reflects whether h improves on the reference set; in the meantime, the reference set is updated every generation. Specifically, noting *wi* the fraction of reference classifiers misclassifying the i-th example, $\mathcal{F}(h)$ (h) is set to the sum of w_i^γ, taken over all examples correctly classified by h. Parameter γ is used to mitigate the influence of noisy examples.

Boosting and Large-Scale Learning

Another key motivation for designing new learning criteria is to yield scalable learning algorithms, coping with giga or terabytes of data (see Sonnenburg et al. 2008).

Song et al. (2003, 2005) presented an elegant genetic programming approach to tackle the intrusion detection challenge (Lippmann et al. 2000); this challenge offers a 500,000 pattern training set, exceeding standard available RAM capacities. The proposed approach relies on the dynamic subset selection method first presented by Gathercole and Ross (1994). The whole dataset is equally and randomly divided into subsets $\mathcal{E}_i$ with same distribution as the whole dataset, where each $\mathcal{E}_i$ fits within the available RAM. Iteratively, some subset $\mathcal{E}_i$ is selected with uniform probability, and loaded in memory; it is used for a number of generations set to $G_{\max} \times Err(i)$ where $G_{\max}$ is the user-supplied maximum number of generations, and $Err(i)$ is the minimum number of patterns in $\mathcal{E}_i$ misclassified the previous time $\mathcal{E}_i$ was considered. Within $\mathcal{E}_i$, a competition is initiated between training patterns to yield a frugal yet challenging assessment of the hypotheses. Specifically, every generation or so, a restricted subset is selected by tournament in $\mathcal{E}_i$, considering both the difficulty of the patterns (the difficulty of pattern $\mathbf{x}_j$ being the number of hypotheses misclassifying $\mathbf{x}_j$ last time $\mathbf{x}_j$ was selected) and its age (the number of generations since $\mathbf{x}_j$ was last selected). With some probability (30 % in the experiments), the tournament returns the pattern with maximum age; otherwise, it returns the pattern with maximum difficulty.

The dynamic selection subset (DSS) heuristics can thus be viewed as a mixture of uniform sampling (modeled by the age-based selection) and boosting (corresponding to the difficulty-based selection). This mixed distribution gets the best of both worlds: it speeds up learning by putting the stress on the most challenging patterns, akin boosting; in the meanwhile, it prevents noisy examples from leading learning astray as the training set always includes a sufficient proportion of uniformly selected examples. The authors report that the approach yields accurate classifiers (though outperformed by the Challenge winning entry), while one trial takes 15 min on a modest laptop computer (1 GHz Pentium, 256 MB RAM).

Gagné et al., aiming at the scalable optimization of SVM kernels, proposed another use of dynamic selection subset in a coevolutionary perspective (Gagné et al. 2006). Specifically, any kernel induces a similarity on the training set

$$s(\mathbf{x}, \mathbf{x}') = 2K(\mathbf{x}, \mathbf{x}') - K(\mathbf{x}, \mathbf{x}) - K(\mathbf{x}', \mathbf{x}')$$

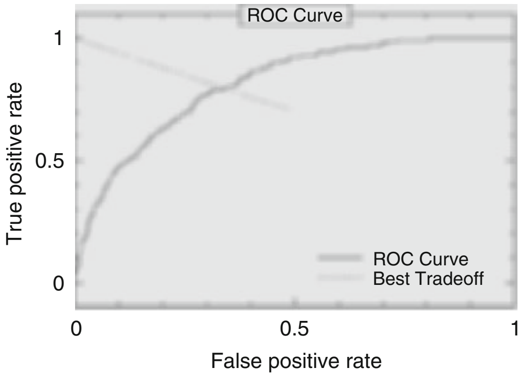
This similarity directly enables the classification of examples along the k-nearest neighbor approach (Duda et al. 2001) (see ▶ Nearest Neighbor), labeling an example after the majority of its neighbors. Inspired from (Gilad-Bachrach et al. 2004), the margin of an example is defined as the rank of its closest neighbor in the same class, minus the rank of its closest neighbor in the other class (the closer a neighbor, the higher its rank is). The larger the margin of an example, the more confident one can be it will be correctly classified; the fitness of the kernel could thus be defined as the sum of the example margins. Computed naively however, this fitness would be quadratic in the size of the training set, hindering the scalability of the approach.

A three-species coevolutionary framework was thus defined. The first species is that of kernels; the second species includes the candidate neighbor instances, referred to as prototypes; the third species includes the training instances, referred to as test cases. Kernels and prototypes undergo a cooperative co-evolution: they cooperate to yield the underlying metric (similarity) and the reference points (prototypes) enabling to classify all training instances. The test cases, in the meanwhile, undergo a competitive coevolution with the other two species: they present the learning process with more and more difficult training examples, aiming at a good coverage of the whole instance space. The approach reportedly yields accurate kernels at a moderate computational cost.

AUC: Area Under the ROC Curve

The misclassification rate criterion is notably ill-suited to problem domains with a minority class. If the goal is to discriminate a rare disease (<1 % of the training set) from a healthy state, the default hypothesis ("everyone is healthy" with 1 % misclassified examples) can hardly be outperformed in terms of predictive accuracy. Standard heuristics accommodating ill-balanced problems involve the oversampling of the minority class, undersampling of the majority class, or cost-sensitive loss function (e.g., misclassifying a healthy person for an ill one costs 1, whereas the opposite costs 100) (Domingos 1999).

Another principled approach is based on the so-called area under the receiver-operating characteristics curve (see ▶ ROC Analysis). Let us consider a continuous hypothesis h, mapping the instance space on the real-value space $\mathbb{R}$. For each threshold τ let the binary classifier h_τ be defined as instance x is positive iff $h(x) > \tau$. To each τ value can be associated the true positive (TP) rate (fraction of ill persons that are correctly classified) and the false positive (FP) rate (fraction of healty persons misclassified as ill ones). In the (FP,TP) plane, the curve drawn as τ varies defines the ROC curve (Fig. 3).



Nonstandard Criteria in Evolutionary Learning, Fig. 3 The receiver operating characteristic (ROC) Curve depicts how the true positive (TP) rate increases vs the false positive (FP) rate. Random guessing corresponds to the diagonal line. The ROC curve is insensitive to ill-balanced distributions as TP and FP rates are normalized

Noting that the ideal classifier lies in the upper left corner (0 % false positive rate, 100 % true positive rate), it comes naturally to optimize the area under the ROC curve. This criterion, also referred to as Wilcoxon rank test, has been intensively studied in both theoretical and algorithmic perspectives (see among many others Cortes and Mohri 2004; Ferri et al. 2002; Joachims 2005; Rosset 2004).

The AUC criterion has been investigated in the EC literature since the 1990s (Fogel et al. 1998), for it defines a combinatorial optimization problem. Considering the search space of real-valued functions, mapping instance space X onto R, the AUC (Wilcoxon) criterion is defined as

$$\mathcal{F}(h) = Pr(h(\mathbf{x}) > h(\mathbf{x}')|y > y')$$

$$\mathcal{F}_e(h) \infty \#\{(\mathbf{x}_i, \mathbf{x}_j) s.t. h(\mathbf{x}_i) > h(\mathbf{x}_j),$$

$$y_i = 1, y_j = 0\}$$

Specifically, hypothesis h is used to rank the instances; any ranking such that all positive instances are ranked before the negative ones gets the optimal AUC. The fitness criterion can be computed with complexity $\mathcal{O}(n \log n)$ where n stands for the number of training instances, by showing that

$$\mathcal{F}_e(h) \infty \sum_{i=1 \ldots n, y_i=1} i \times \text{rank}(i)$$

Interestingly, the optimization of the AUC criterion can be dealt with in the SVM framework, as shown by Joachims (2005), replacing class constraints by inegality constraints (Fig. 2):

$$y_i(< w, \mathbf{x}_i > +b) \geq 1 \; i = 1 \ldots n$$
$$\rightarrow \; < w, \mathbf{x}_i - \mathbf{x}_j > \geq 1 \; i, j = 1 \ldots n, s.t. y_i > y_i$$

In practice, the quadratic optimization process introduces gradually the violated constraints only, to avoid dealing with a quadratic number of constraints.

The flexibility of EC can still allow for more specific and application-driven interpretation of the AUC criterion. Typically in medical applications, the physician is most interested in the beginning of the AUC curve, trying to find a threshold τ retrieving a high fraction of ill patients for a very low false positive rate. The same situation occurs in customer relationship management, replacing positive cases by potential churners. The AUC criterion can be easily adapted to minimize the number of false positive within the top k-ranked individuals, as shown by Mozer et al. (2001).

In a statistical perspective however (and contrarily to a common practice in the ML and data mining communities), it has been argued that selecting a classifier based on its AUC was not appropriate (Hand 2009). The objection is that the AUC maximization yields the best hypothesis *under a uniform distribution of the misclassification costs*, whereas hypothesis h is used with a specific threshold τ, corresponding to a particular point of the ROC curve (Fig. 3).

Still, ROC curves convey very clear intuitions about the trade-off between TP and FP rates; analogous to a Pareto front, they enable one to select a posteriori the best trade-off according to a one's implicit preferences. An interesting approach along these lines has been investigated by Suttorp and Igel (2006) to learn SVMs, using a multi-objective optimization setting to simultaneously minimize the FP rate, and maximize the TP rate, and maximize the number of support vectors.

The last objective actually corresponds to a regularization term: the empirical error plus the number of support vectors upper-bounds the so-called leave-one-out error (when the number of folds in cross-fold validation is set to the number of examples), since the hypothesis is not modified when removing a non-support vectors. (see Zhang (2003) for more detail).

Conclusions

Unsurprisingly, the bottom line of evolutionary learning matches that of EC: any effort to customize the fitness function is highly rewarded; a good knowledge of the domain application enables to choose appropriate, frugal yet effective, search space and variation operators.

Another message concerns the validation of the proposed approaches. In early decades, hypotheses were assessed from their training error, with poor applicative relevance due to overfitting. Better practices are now widely used (e.g., training, validation, and test sets); as advocated by Dietterich (1998), good practices are based on cross-validation. Taking into account early remarks about the University of California Irvine (UCI) repository (Holte 1993), experimental validation should consider actually challenging problems.

Due to space limitations, this entry has excluded some nice and elegant work at the crossroad of machine learning and evolutionary computation, among others, interactive optimization and modelisation of the user's preferences (Llorà et al. 2005), interactive feature construction (Krawiec and Bhanu 2007;

Venturini et al. 1997), or ML-based heuristics for noisy optimization (Heidrich-Meisner and Igel 2009).

Recommended Reading

Amit Y, Geman D, Wilder K (1997) Joint induction of shape features and tree classifiers. IEEE Trans Pattern Anal Mach Intell 19(11):1300–1305

Banzhaf W, Langdon WB (2002) Some considerations on the reason for bloat. Genet Progr Evolvable Mach 3(1):81–91

ben-David S, von Luxburg U, Shawe-Taylor J, Tishby N (eds) (2005) Theoretical foundations of clustering. In: NIPS workshop

bishop C (2006) Pattern recognition and machine learning. Springer, New York

Blickle T (1996) evolving compact solutions in genetic programming: a case study. In: Voigt H-M et al (eds) Proceedings of the 4th international inference on parallel problem solving from nature. Lecture notes in computer science, vol 1141. Springer, Berlin, pp 564–573

boser B, Guyon I, Vapnik V (1992) A training algorithm for optimal margin classifiers. In: Proceedings of the 5th annual ACM conference on computational learning theory (COLT'92), Pittsburgh, pp 144–152

Breiman L (1998) Arcing classifiers. Ann Stat 26(3):801–845

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Chandra A, Yao X (2006a) Ensemble learning using multi-objective evolutionary algorithms. J Math Model Algorithm 5(4):417–425

Chandra A, Yao X (2006b) Evolving hybrid ensembles of learning machines for better generalisation. Neurocomputing 69:686–700

Cortes C, Vapnik VN (1995) Support-vector networks. Mach Learn 20:273–297

Cortes C, Mohri M (2004) Confidence intervals for the area under the ROC curve. Adv Neural Inf Process Syst NIPS 17

Cristianini N, Shawe-Taylor J (2000) An introduction to support vector machines and other kernel-based learning methods. Cambridge University Press, Cambridge

Hand DJ (2009) Measuring classifier performance: a coherent alternative to the area under the ROC curve. Mach Learn 77(1):103–123. http://dx.doi.org/10.1007/S10994-009-5119-5. DBLP http://dblp.uni-trier.de

Deb K, Agrawal S, Pratab A, Meyarivan T (2000) A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: Schoenauer M et al (eds) Proceedings of the parallel problem solving from nature VI conference, Paris. Lecture notes in computer science, vol 1917. Springer, pp 849–858

Dietterich TG (1998) Approximate statistical tests for comparing supervised classification learning algorithms. Neural Comput 10:1895–1923

Dietterich T (2000) Ensemble methods in machine learning. In: Kittler J, Roli F (eds) First international workshop on multiple classifier systems. Springer, Berlin, pp 1–15

Domingos P (1999) Meta-cost: a general method for making classifiers cost sensitive. In: Proceedings of the 5th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, San Diego, pp 155–164

Duda RO, Hart PE, Stork DG (2001) Pattern classification, 2nd ed. Wiley, New York

Ferri C, Flach PA, Hernndez-Orallo J (2002) Learning decision trees using the area under the ROC curve. In: Sammut C, Hoffman AG (eds) Proceedings of the nineteenth international conference on machine learning (ICML 2002). Morgan Kaufmann, pp 179–186

Fogel DB, Wasson EC, Boughton EM, Porto VW, Angeline PJ (1998) Linear and neural models for classifying breast cancer. IEEE Trans Med Imag 17(3):485–488

Freund Y, Shapire RE (1996) Experiments with a new boosting algorithm. In: Saitta L (ed) Proceedings of the thirteenth international conference on machine learning (ICML 1996). Morgan Kaufmann, Bari, pp 148–156

Friedrichs F, Igel C (2005) Evolutionary tuning of multiple SVM parameters. Neurocomputing 64(C):107–117

Gagné C, Schoenauer M, Sebag M, Tomassini M (2006) Genetic programming for kernel-based learning with co-evolving subsets selection. In: Runarsson TP, Beyer H-G, Burke EK, Merelo Guervós JJ, Whitley LD, Yao X (eds) Parallel problem solving from nature – PPSN IX. Lecture notes in computer science, vol 4193, pp 1008–1017. Springer

Gagné C, Sebag M, Schoenauer M, Tomassini M (2007) Ensemble learning for free with evolutionary algorithms? In: Lipson H (ed) Genetic and evolutionary computation conference (GECCO 2007). ACM, pp 1782–1789

Gathercole C, Ross P (1994) Dynamic training subset selection for supervised learning in genetic programming. In: Parallel problem solving from nature – PPSN III. Lecture notes in computer science, vol 866. Springer, pp 312–321

Gelly S, Teytaud O, Bredeche N, Schoenauer M (2006) Universal consistency and bloat in GP: some theoretical considerations about genetic programming from a statistical learning theory viewpoint. Revue d'Intell Artif 20(6):805–827

Gilad-Bachrach R, Navot A, Tishby N (2004) Margin based feature selection – theory and algorithms. In: Proceedings of the twenty-first international conference on machine learning (ICML 2009), Montreal. ACM Press, p 43

Han J, Kamber M (2000) Data mining: concepts and techniques. Morgan Kaufmann, New York

Hastie T, Rosset S, Tibshirani R, Zhu J (2004) The entire regularization path for the support vector machine. Adv Neural Inf Process Syst NIPS 17

Heidrich-Meisner V, Igel C (2009) Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search. Proceedings of the twenty-sixth international conference on machine learning (ICML 2009), Montreal. ACM, pp 401–408

Hillis WD (1990) Co-evolving parasites improve simulated evolution as an optimization procedure. Phys D 42:228–234

Holland J (1986) Escaping brittleness: the possibilities of general purpose learning algorithms applied to parallel rule-based systems. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning: an artificial intelligence approach, vol 2. Morgan Kaufmann, Los Altos, pp 593–623

Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor

Holte RC (1993) Very simple classification rules perform well on most commonly used datasets. Mach Learn 11:63–90

Monirul Islam M, Yao X (2008) Evolving artificial neural network ensembles. In: Fulcher J, Jain LC (eds) Computational intelligence: a compendium. Studies in computational intelligence, vol 115. Springer, pp 851–880

Joachims T (2005) A support vector method for multivariate performance measures. In: De Raedt L, Wrobel S (eds) Proceedings of the twenty-second international conference on machine learning (ICML 2009), Montreal. ACM international conference proceeding series, vol 119. ACM, pp 377–384

Jong K, Marchiori E, Sebag M (2004) Ensemble learning with evolutionary computation: application to feature ranking. In: Yao X et al (eds) Parallel problem solving from nature – PPSN VIII. Lecture notes in computer science, vol 3242. Springer, pp 1133–1142

Miikkulainen R, Stanley KO, Bryant BD (2003) Evolving adaptive neural networks with and without adaptive synapses. Evol Comput 4:2557–2564

Krawiec K, Bhanu B (2007) Visual learning by evolutionary and coevolutionary feature synthesis. IEEE Trans Evol Comput 11(5):635–650

Lippmann R, Haines JW, Fried DJ, Korba J, Das K (2000) Analysis and results of the 1999 DARPA online intrusion detection evaluation. In: Debar H, Mé L, Wu SF (eds) Recent advances in intrusion detection. Lecture notes in computer science, vol 1907. Springer, Berlin, pp 162–182

Liu Y, Yao X, Higuchi T (2000) Evolutionary ensembles with negative correlation learning. IEEE Trans Evol Comput 4(4):380–387

Llorà X, Sastry K, Goldberg DE, Gupta A, Lakshmi L (2005) Combating user fatigue in IGAS: partial ordering, support vector machines, and synthetic fitness. In: Beyer H-G, O'Reilly U-M (eds) Genetic and evolutionary computation conference (GECCO 05). ACM, New York, pp 1363–1370

Margineantu D, Dietterich TG (1997) Pruning adaptive boosting. In: Proceedings of the fourteenth international conference on machine learning (ICML 1996), Bari. Morgan Kaufmann, pp 211–218

Mierswa I (2006) Evolutionary learning with kernels: a generic solution for large margin problems. In: Cattolico M (ed) Genetic and evolutionary computation conference (GECCO 06). ACM, New York, pp 1553–1560

Mierswa I (2007) Controlling overfitting with multiobjective support vector machines. In: Lipson H (ed) Genetic and evolutionary computation conference (GECCO 07), Philadelphia, pp 1830–1837

Mozer MC, Dodier R, Colagrosso MC, Guerra-Salcedo C, Wolniewicz R (2001) Prodding the ROC curve: constrained optimization of classifier performance. Adv Neural Inf Process Syst NIPS. MIT Press

Platt J (1999) Fast training of support vector machines using sequential minimal optimization. In: Schölkopf B et al (eds) Advances in kernel methods – support vector learning. Morgan Kaufmann

Poli R (2008) Genetic programming theory. In: Ryan C, Keijzer M (eds) Genetic and evolutionary computation conference (GECCO 2008), Atlanta (Companion). ACM, pp 2559–2588

Rosset S (2004) Model selection via the AUC. In: Proceedings of the twenty-first international conference on machine learning (ICML 2009), Montreal. ACM international conference proceeding series, vol 69. ACM

Rumelhart DE, McClelland JL (1990) Parallel distributed processing. MIT Press, Cambridge

Schapire RE (1990) The strength of weak learnability. Mach Learn 5:197

Schoenauer M, Xanthakis S (1993) Constrained GA optimization. In: Forrest S (ed) Proceedings of the 5th international conference on genetic algorithms. Morgan Kaufmann, San Mateo, pp 573–580

Schölkopf B, Burges C, Smola A (1998) Advances in Kernel methods: support vector machines. MIT Press, Cambridge

Song D, Heywood MI, Nur Zincir-heywood A (2003) A linear genetic programming approach to intrusion detection. In: Proceedings of the genetic and evolutionary computation conference (GECCO). Lecture notes in computer science, vol 2724. Springer, Berlin/New York, pp 2325–2336

Song D, Heywood MI, Nur Zincir-Heywood A (2005) Training genetic programming on half a million patterns: an example from anomaly detection. IEEE Trans Evol Comput 9(3):225–239

Sonnenburg S, Franc V, Yom-Tov E, Sebag M (eds) (2008) Large scale machine learning challenge. In: ICML workshop, Helsinki

Sutton RS, Barto AG (1998) Reinforcement learning. MIT Press, Cambridge

Suttorp T, Igel C (2006) Multi-objective optimization of support vector machines. In: Jin Y (ed) Multi-objective machine learning. Studies in computational intelligence, vol 16. Springer, Berlin, pp 199–220

Tibshirani R (1996) Regression shrinkage and selection via the lasso. R Stat Soc B 58(1):267–288

Vapnik VN (1995) The nature of statistical learning. Springer, New York

Venturini G, Slimane M, Morin F, Asselin de Beauville JP (1997) On using interactive genetic algorithms for knowledge discovery in databases. In: Bäck Th (ed) International conference on genetic algorithms (ICGA). Morgan Kaufmann, pp 696–703

Zhang T (2003) Leave-one-out bounds for kernel methods. Neural Comput 15(6):1397–1437

Nonstationary Kernels

▶ Local Distance Metric Adaptation
▶ Locally Weighted Regression for Control

Normal Distribution

▶ Gaussian Distribution

NP-Completeness

Definition

A *decision problem* consists in identifying symbol strings, presented as inputs, that have some specified property. The output consists in a yes/no or 0/1 answer. A decision problem belongs to the class P if there exists an algorithm, that is, a deterministic procedure, for deciding any instance of the problem in a length of time bounded by a polynomial function of the length of the input.

A decision problem is in the class NP if it is possible for every yes-instance of the problem to verify in polynomial time, after having been supplied with a polynomial-length *witness*, that the instance is indeed of the desired property.

An example is the problem to answer the question for two given numbers n and m whether n has a divisor d strictly between m and n. This problem is in NP: if the answer is positive, then such a divisor d will be a witness, since it can be easily checked that d lies between the required bounds, and that n is indeed divisible by d. However, it is not known whether this decision problem is in P or not, as it may not be easy to find a suitable divisor d, even if one exists.

The class of *NP-complete* decision problems contains such problems in NP for which if some algorithm decides it, then every problem in NP can be decided in polynomial time. A theorem of Stephen Cook and Leonid Levin states that such decision problems exist. Several decision problems of this class are problems on ▶ graphs.

Recommended Reading

Cook S (1971) The complexity of theorem proving procedures. In: Proceedings of the third annual ACM symposium on theory of computing, pp 151–158

Levin L (1973) Universal'nye pereborne zadachi. Probl Peredachi Inf 9(3):265–266

English translation, Universal Search Problems, in Trakhtenbrot BA (1984) A survey of Russian approaches to Perebor (Brute-Force Searches) algorithms. Ann Hist Comput 6(4): 384–400

Numeric Attribute

Synonyms

Quantitative attribute

Definition

Numeric attributes are attributes that are numerical in nature. Their values can be ranked in order and can be subjected to meaningful arithmetic operations. See ▶ Attribute and ▶ Measurement Scales.

O

Object

▶ Instance

Object Consolidation

▶ Entity Resolution

Object Identification

▶ Record Linkage

Object Matching

▶ Record Linkage

Object Space

▶ Instance Space

Objective Function

▶ Partitional Clustering

Observation Language

Hendrik Blockeel
Katholieke Universiteit Leuven, Heverlee, Leuven, Belgium
Leiden Institute of Advanced Computer Science, Heverlee, Belgium

Synonyms

Instance language

Definition

The *observation language* used by a machine learning system is the language in which the observations it learns from are described.

Motivation and Background

Most machine learning algorithms can be seen as a procedure for deriving one or more hypotheses from a set of observations. Both the input (the observations) and the output (the hypotheses) need to be described in some particular language and this language is called the observation language or the ▶ Hypothesis Language respectively. These terms are mostly used in the context of symbolic learning, where these languages are often more complex than in subsymbolic or statistical learning.

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

The following sections describe some of the key observation languages.

Attribute-Value Learning

Probably the most used setting in machine learning is the *attribute-value* setting (see ▶ Attribute-Value Learning). Here, an example (observation) is described by a fixed set of attributes, each of which is given a value from the domain of the attribute. Such an observation is often called a vector or, in relational database terminology, a tuple. The attributes are usually atomic (i.e., not decomposable in component values) and single-valued (i.e., an attribute has only one value, not a set of values). So we have an instance space (or space of observations)

$$\mathcal{O} = A_1 \times \cdots \times A_n,$$

elements of which are denoted using an observation language that typically has the same structure:

$$\mathcal{L}_O = \mathcal{L}_{A_1} \times \cdots \times \mathcal{L}_{A_n},$$

(the language contains tuples of objects that represent the attribute values).

The attribute-value framework easily allows for both supervised and unsupervised learning; in the supervised learning setting, the label of an instance is simply included as an attribute in the tuple, where as for unsupervised learning, it is excluded.

The attribute-value setting assumes that all instances can be represented using the same fixed set of attributes. When instances can be of different types or are variable-sized (e.g., when an instance is set-valued), this assumption may not hold, and more powerful languages may have to be used instead.

Learning from Graphs, Trees, or Sequences

We here consider the case in which a single instance is a graph, or a node in a graph. Note that trees and sequences are special cases of graphs.

A graph is defined as a pair (V, E), where V is a set of vertices and E a set of edges each edge being a pair of vertices. If the pair is ordered, the graph is directed; otherwise it is undirected. For simplicity, we restrict ourselves to undirected graphs.

A graph can, in practice, not be encoded in attribute-value format without the loss of information. That is, one could use a number of properties of graphs as attributes in the encoding, but several graphs may then still map onto the same representation, which implies loss of information. In theory, one could imagine defining a total order on (certain classes of) graphs and representing each graph by its rank in that order (which is a single numerical attribute), thus representing graphs as numbers without loss of information; but then it is not obvious how to map patterns in this numerical representation to patterns in the original representation. No such approaches have been proposed till now.

Describing the instance space is more difficult here than in the attribute value case. Consider a task of graph classification, where in observations are of the form (G, y) with G a graph and y a value for a target attribute Y. Then we can define the instance space as

$$\mathcal{O} = \{(V, E) | V \subseteq \mathbf{N} \wedge E \subseteq V^2\} \times Y,$$

where $\mathbf{N}$ is the set of all natural numbers. (For each graph, there exists a graph defined over $\mathbf{N}$ that is isomorphic with it, so $\mathcal{O}$ contains all possible graphs up to isomorphism.)

A straightforward observation language in the case of graph classification is then

$$\begin{aligned}&\{(G, y) | G\\ &\quad = (V, E) \wedge V \subseteq \mathcal{L}_V \wedge E \subseteq V^2 \wedge y \in Y\},\end{aligned}$$

where $\mathcal{L}_V$ is some alphabet for representing nodes.

In learning from graphs, there are essentially two settings: those where a prediction is made for entire graphs, and those where a prediction is made for single nodes in a graph. In the first case, observations are of the form (G, y), where

Anne	1997
Bernard	1999
Celine	1996
Daniel	1999
Elisa	1997
Fabian	1999

Anne	Algebra	1998	A
Anne	Calculus	1998	B
Bernard	Databases	2000	A
Celine	Biology	1999	B
Celine	Databases	2000	B
Celine	Calculus	1998	A

Algebra
Biology
Calculus
Databases

Adams	Algebra	1998
Adams	Calculus	1999
Baeck	Biology	1999
Cools	Calculus	1998
Cools	Databases	1999

Adams
Baeck
Cools

Observation Language, Fig. 1 A small database of students

as, in the second case, they are of the form (G, v, y), where $G = (V, E)$ and $v \in V$. That is, a node is given together with the graph in which it occurs (its "environment"), and a prediction is to be made for this specific node, using the information about its environment.

In many cases, the set of observations one learns from is of the form (G, v_i, y_i), where each instance is a different node of exactly the same graph G. This is the case when, for instance, classifying web pages, we take the whole web as their environment.

In a labeled graph, labels are associated with each node or edge. Often these are assumed atomic, being elements of a finite alphabet or real numbers, but they can also be vectors of reals.

Relational Learning

In ▸ relational learning, it is assumed that relationships may exist between different instances of the instance space, or an instance may internally consist of multiple objects among which relationships exist.

This essentially corresponds to learning from graphs, except that in a graph only one binary relation exists (the edges E), whereas here there may be multiple relations and they may be non binary. The expressiveness of the two settings is the same, however, as any relation can be represented using only binary relations.

In the attribute-value setting, one typically uses one table where each tuple represents all the relevant information for one observation. In the relational setting, there may be multiple tables, and information on a single instance is contained in multiple tuples, possibly belonging to multiple relations.

Example 1 Assume we have a database about students, courses, and professors (see Fig. 1). We can define a single observation as all the information relevant to one student, that is: the name, year of entrance, etc. of the student and also the courses they take and the professors teaching these courses.

The most obvious link to the graph representation is as follows: create one node for each tuple, labeled with that tuple, and create a link between two nodes if the corresponding tuples are connected by a foreign key relationship.

Defining a single observation as a set of tuples that are connected through foreign keys in the database corresponds to representing each observation (G, v, y) as (G', v, y), where G' is the connected component of G that contains v. The actual links are usually not explicitly written in this representation, as they are implicit: there is an edge between two tuples if they have the same value for a foreign key attribute.

Inductive Logic Programming

In ▸ inductive logic programming, a language based on first order logic is used to represent the observations. Typically, an observation is then represented by a *ground fact*, which basically corresponds to a single tuple in a relational database. In some settings an observation is represented by an *interpretation*, a set of ground facts, which corresponds to the set of tuples mentioned in the previous subsection.

While the target variable can always be represented as an additional attribute, ILP systems often learn from examples and counterexamples of a concept. The target variable is then implicit: it is true or false depending on whether the example is in the positive or negative set, but it is not explicitly included in the fact.

Typical for the inductive logic programming setting is that the input of a system may contain, besides the observations, background knowledge about the application domain. The advantage of the ILP setting is that no separate language is needed for such background knowledge: the same first order logic-based language can be used for representing the observations as well as the background knowledge.

Example 2 *Take the following small dataset*:

```
sibling(bart,lisa).
sibling(lisa,bart).
:- sibling(bart, bart).
:- sibling(lisa, lisa).
father(homer, bart).
mother(marge, bart).
father(homer, lisa).
mother(marge, lisa).
```

There are positive and negative (preceded by :-) examples of the Sibling relation. The following hypothesis might be learned:

```
sibling(X,Y) :- father(Z,X),
father(Z,Y), X ≠ Y.
sibling(X,Y) :- mother(Z,X),
mother(Z,Y), X ≠ Y.
```

If the following clauses as included as background knowledge:

```
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

then the same ILP system might learn the following more compact definition:

```
sibling(X,Y) :- parent(Z,X),
parent(Z,Y), X ≠ Y.
```

Further Reading

Most of the literature on hypothesis and observation languages is found in the area of inductive logic programming. Excellent starting points to become familiar with this field are *Relational Data Mining* by Džeroski and Lavraè (2001) and *Logical and Relational Learning* by De Raedt (2008).

De Raedt (1998) compares a number of different observation and hypothesis languages with respect to their expressiveness, and indicates relationships between them.

Cross-References

▸ Hypothesis Language
▸ Inductive Logic Programming
▸ Relational Learning

Recommended Reading

De Raedt L (1998) Attribute-value learning versus inductive logic programming: the missing links (extended abstract). In: Page D (ed) Proceedings of the eighth international conference on inductive logic programming. Lecture notes in artificial intelligence, vol 1446. Springer, Berlin, pp 1–8

De Raedt L (2008) Logical and relational learning. Springer, Berlin

Džeroski S, Lavraè N (eds) (2001) Relational data mining. Springer, Berlin.

Occam's Razor

Geoffrey I. Webb
Faculty of Information Technology, Monash University, Victoria, Australia

Synonyms

Ockham's Razor

Definition

Occam's Razor is the maxim that "entities are not to be multiplied beyond necessity," or as it is often interpreted in the modern context "of two hypotheses H and H', both of which explain E, the simpler is to be preferred" (Good 1977)

Motivation and Background

Most attempts to learn a model from data confront the problem that there will be many models that are consistent with the data. In order to learn a single model, a choice must be made between the available models. The factors taken into account by a learner in choosing between models are called its learning biases (Mitchell 1980). A preference for simple models is a common learning bias and is embodied in many learning techniques including pruning, minimum message length, and minimum description length. Regularization is also sometimes viewed as an application of Occam's razor.

Occam's razor is an imperative, rather than a proposition. That is, it is neither true nor false. Rather, it is a call to act in a particular way without making any claim about the consequences of doing so. In machine learning the so-called *Occam thesis* is sometimes assumed that: given a choice between two plausible classifiers that perform identically on the training set, the simpler classifier is expected to classify correctly more objects outside the training set. (Webb 1996)

While there are many practical advantages in having a learning bias toward simple models, there remains controversy as to whether the Occam thesis is true (Webb 1996; Domingos 1999; Blumer et al. 1987).

Cross-References

▶ Learning Bias
▶ Language Bias
▶ Minimum Description Length Principle
▶ Minimum Message Length
▶ Pruning
▶ Regularization

Recommended Reading

Blumer A, Ehrenfeucht A, Haussler D, Warmuth MK (1987) Occam's razor. Inf Process Lett 24(6):377–380

Domingos P (1999) The role of Occam's razor in knowledge discovery. Data Min Knowl Discov 3(4):409–425

Good IJ (1977) Explicativity: a mathematical theory of explanation with statistical applications. Proc R Soc Lond Ser A 354:303–330

Mitchell TM (1980) The need for biases in learning generalizations. Department of computer science, Technical report CBM-TR-117, Rutgers University

Webb GI (1996) Further experimental evidence against the utility Of occams razor. J Artif Intell Res 4:397–417. AAAI Press, Menlo Park

Ockham's Razor

▶ Occam's Razor

Offline Learning

▶ Batch Learning

One-Against-All Training

▶ Class Binarization

One-Against-One Training

▶ Class Binarization

1-Norm Distance

▶ Manhattan Distance

One-Step Reinforcement Learning

▶ Associative Reinforcement Learning

Online Controlled Experiments and A/B Testing

Ron Kohavi[1] and Roger Longbotham[2]
[1]Application Services Group, Microsoft, Bellevue, WA, USA
[2]Data and Decision Sciences Group, Microsoft, Redmond, WA, USA

Abstract

The Internet connectivity of client software (e.g., apps running on phones and PCs), websites, and online services provide an unprecedented opportunity to evaluate ideas quickly using controlled experiments, also called A/B tests, split tests, randomized experiments, control/treatment tests, and online field experiments. Unlike most data mining techniques for finding correlational patterns, controlled experiments allow establishing a causal relationship with high probability. Experimenters can utilize the scientific method to form a hypothesis of the form "If a specific change is introduced, will it improve key metrics?" and evaluate it with real users.

The theory of a controlled experiment dates back to Sir Ronald A. Fisher's experiments at the Rothamsted Agricultural Experimental Station in England in the 1920s, and the topic of offline experiments is well developed in Statistics (Box et al., Statistics for experimenters: design, innovation, and discovery. Wiley, Hoboken, 2005). Online-controlled experiments started to be used in the late 1990s with the growth of the Internet. Today, many large sites, including Amazon, Bing, Facebook, Google, LinkedIn, and Yahoo!, run thousands to tens of thousands of experiments each year testing user interface (UI) changes, enhancements to algorithms (search, ads, personalization, recommendation, etc.), changes to apps, content management system, etc. Online-controlled experiments are now considered an indispensable tool, and their use is growing for startups and smaller websites. Controlled experiments are especially useful in combination with Agile software development (Martin, Clean code: a handbook of Agile software craftsmanship. Prentice Hall, Upper Saddle River, 2008; Rubin, Essential scrum: a practical guide to the most popular Agile process. Addison-Wesley Professional, Upper Saddle River, 2012), Steve Blank's Customer Development process (Blank, The four steps to the epiphany: successful strategies for products that win. Cafepress.com., 2005), and MVPs (minimum viable products) popularized by Eric Ries's Lean Startup (Ries, The lean startup: how today's entrepreneurs use continuous innovation to create radically successful businesses. Crown Business, New York, 2011).

Synonyms

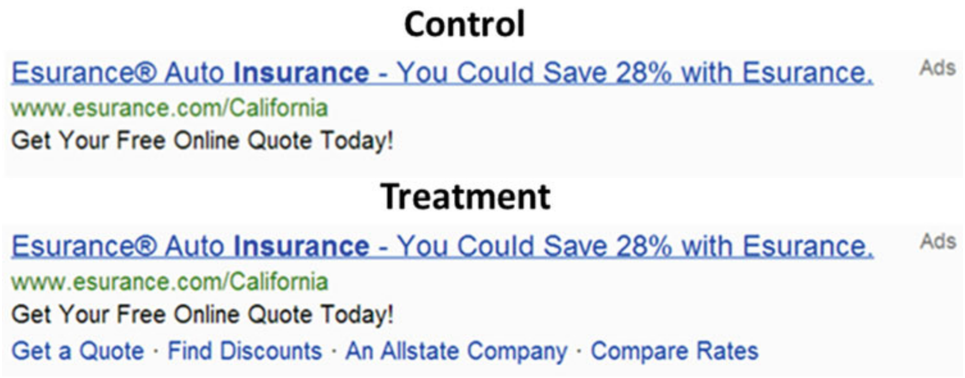
A/B Testing; Randomized Experiments; Split Tests

Motivation and Background

Many good resources are available with motivation and explanations about online-controlled experiments (Siroker and Koomen 2013; Goward 2012; McFarland 2012b; Schrage 2014; Kohavi et al. 2009, 2014, 2013).

We provide a motivating visual example of a controlled experiment that ran at Microsoft's Bing. The team wanted to add a feature allowing advertisers to provide links to the target site. The rationale is that this will improve ads' quality by giving users more information about what the advertiser's site provides and allows users to directly navigate to the subcategory matching their intent. Visuals of the existing ads layout (control) and the new ads layout (treatment) with site links added are shown in Fig. 1.

In a controlled experiment, users are randomly split between the variants (e.g., the two different ads layouts) in a persistent manner (a user receives the same experience in multiple visits). Their interactions with the site are instrumented



Control

Esurance® Auto Insurance - You Could Save 28% with Esurance.
Ads
www.esurance.com/California
Get Your Free Online Quote Today!

Treatment

Esurance® Auto Insurance - You Could Save 28% with Esurance.
Ads
www.esurance.com/California
Get Your Free Online Quote Today!
Get a Quote · Find Discounts · An Allstate Company · Compare Rates

Online Controlled Experiments and A/B Testing, Fig. 1 Ads with site link experiment. Treatment (*bottom*) has site links. The difference might not be obvious at first but it is worth tens of millions of dollars

and key metrics computed. In this experiment, the Overall Evaluation Criterion (OEC) was simple: increasing average revenue per user to Bing without degrading key user engagement metrics. Results showed that the newly added site links increased revenue, but also degraded user metrics and page load time, likely because of increased vertical space usage. Even offsetting the space by lowering the average number of mainline ads shown per query, this feature improved revenue by tens of millions of dollars per year with neutral user impact, resulting in extremely high ROI (return on investment).

Running online-controlled experiments is not applicable for every organization. We begin with key tenets, or assumptions, an organization needs to adopt (Kohavi et al. 2013).

Tenet 1: The Organization Wants to Make Data-Driven Decisions and Has Formalized the Overall Evaluation Criterion (OEC)

You will rarely hear someone at the head of an organization say that they don't want to be data-driven, but measuring the incremental benefit to users from new features has costs, and objective measurements typically show that progress is not as rosy as initially envisioned. In any organization, there are many important metrics reflecting revenue, costs, customer satisfaction, loyalty, etc., and very frequently an experiment will improve one but hurt another of these metrics. Having a single metric, which we call the Overall Evaluation Criterion, or OEC, that is at a higher level than these and incorporates the tradeoff among them is essential for organizational decision-making.

An OEC has to be defined, and it should be measurable over relatively short durations (e.g., 2 weeks). The hard part is finding metrics that are measurable in the short-term that are predictive of long term goals. For example, "profit" is not a good OEC, as short-term theatrics (e.g., raising prices) can increase short-term profit, but hurt it in the long run. As shown in *Trustworthy Online Controlled Experiments: Five Puzzling Outcomes Explained* (Kohavi et al. 2012), market share can be a long-term goal, but it is a terrible short-term criterion: making a search engine worse forces people to issue more queries to find an answer, but, like hiking prices, users will find better alternatives long-term. Sessions per user, or repeat visits, is a much better OEC for a search engine. Thinking of the drivers of lifetime value can lead to a strategically powerful OEC (Kohavi et al. 2009). We cannot overemphasize the importance of coming up with a good OEC that the organization can align behind.

Tenet 2: Controlled Experiments Can Be Run and Their Results Are Trustworthy

Not every decision can be made with the scientific rigor of a controlled experiment. For example, you cannot run a controlled experiment on the possible acquisition of one company by another. Hardware devices may have long lead times for manufacturing, and modifications are hard, so controlled experiments with actual users are hard to run on a new phone or tablet. For customer-facing websites and services, changes are easy to make through software, and running controlled experiments is relatively easy.

Assuming you can run controlled experiments, it is important to ensure their trustworthiness. When running online experiments, getting numbers is easy; getting numbers you can trust is hard, and we have had our share of pitfalls and puzzling results (Kohavi et al. 2012, 2010; Kohavi and Longbotham 2010).

Tenet 3: We Are Poor at Assessing the Value of Ideas

Features are built because teams believe they are useful, yet in many domains, most ideas fail to improve key metrics. Only one third of the ideas tested on the Experimentation Platform at Microsoft improved the metric(s) they were designed to improve (Kohavi et al. 2009). Success is even harder to find in well-optimized domains like Bing. Jim Manzi (2012) wrote that at Google, only "about 10 percent of these [controlled experiments, were] leading to business changes." Avinash Kaushik wrote in his Experimentation and Testing primer (Kaushik 2006) that "80 % of the time you/we are wrong about what a customer wants." Mike Moran (2007, 240) wrote that Netflix considers 90 % of what they try to be wrong. Regis Hadiaris from Quicken Loans wrote that "in the five years I've been running tests, I'm only about as correct in guessing the results as a major league baseball player is in hitting the ball. That's right - I've been doing this for 5 years, and I can only "guess" the outcome of a test about 33 % of the time!" (Moran 2008). Dan McKinley at Etsy wrote (McKinley 2013) "nearly everything fails" and "it's been humbling to realize how rare it is for them [features] to succeed on the first attempt. I strongly suspect that this experience is universal, but it is not universally recognized or acknowledged." Finally, Colin McFarland wrote in the book *Experiment!* (McFarland 2012b, 20) "No matter how much you think it's a no-brainer, how much research you've done, or how many competitors are doing it, sometimes, more often than you might think, experiment ideas simply fail."

Not every domain has such poor statistics, but most who have run controlled experiments in customer-facing websites and applications have experienced this humbling reality: we are poor at assessing the value of ideas, and that is the greatest motivation for getting an objective assessment of features using controlled experiments.

Structure of an Experimentation System

Elements of an Experimentation System

The simplest experimental setup is to evaluate a factor with two levels, a control (version A) and a treatment (version B). The control is normally the default version, and the treatment is the change that is tested. Such a setup is commonly called an A/B test. It is commonly extended by having several levels, often referred to as A/B/n split tests. An experiment with multiple factors is referred to as multivariable (or multivariate).

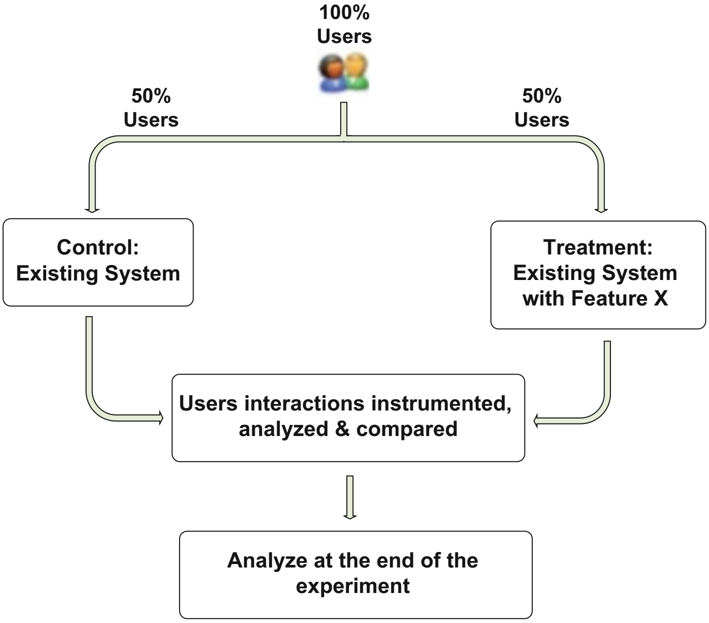
Figure 2 shows the high-level structure of an A/B experiment. In practice, one can assign any percentages to the treatment and control, but 50 % provides the experiment the maximum statistical power, and we recommend maximally powering the experiments after a ramp-up period at smaller percentages to check for egregious errors.

In a general sense, the analysis will test if the statistical distribution of the treatment is different from that of the control. In practice, the most common test is whether the two means are equal or not. For this case, the effect of version B (or treatment effect) is defined to be

$$E(B) = \bar{X}_B - \bar{X}_A \qquad (1)$$

where X is a metric of interest and $\bar{X}_B$ is the mean for variant B. However, for interpretability, the percent change is normally reported with a suitable (e.g., 95 %) confidence interval. See, for example, Kohavi et al. (2009).

Control of extraneous factors and randomization are two essential elements of any experimentation system. Any factor that may affect



Online Controlled Experiments and A/B Testing, Fig. 2 High-level structure of an online experiment

an online metric is either a test factor (one you intentionally vary to determine its effect) or a non-test factor. Non-test factors could either be held fixed, blocked, or randomized. Holding a factor fixed can impact external validity and is thus not recommended. For example, if weekend days are known to be different from week days, you could run the experiment only on weekdays (or weekends), but it would be better to have complete weeks in the experiment for better external validity. Blocking (e.g., pairing) can reduce the variance relative to randomization and is recommended when experimentation units in each block are more homogenous than between blocks. For example, if the randomization unit is a user page view, then blocking by weekend/weekday can reduce the variance of the effect size, leading to higher sensitivity. Time is a critical non-test factor, and because many external factors vary with time, it is important to randomize over time by running the control and treatment(s) concurrently with a fixed percentage to each throughout the experiment. (If the relative percentage changes, you will be subject to Simpson's paradox (Malinas and Bigelow 2009; Kohavi and Longbotham 2010).) Controlling a non-test factor assures it will have equal influence on the control and treatment, hence not affecting the estimate of the treatment effect.

Experimentation Architecture Alternatives

Controlled experiments on the web: survey and practical guide (Kohavi et al. 2009) provides a review of many architecture alternatives. The main three components of an experimentation capability involve the randomization algorithm, the assignment method (i.e., how the randomly assigned experimental units are given the variants), and the data path (which captures raw observation data and processes it). Tang et al. (2010) give a detailed view of the infrastructure for experiments as carried out by Google.

To validate an experimentation system, we recommend that A/A tests be run regularly to test that the experimental setup and randomization mechanism is working properly. An A/A test, sometimes called a null test (Peterson 2004), exercises the experimentation system, assigning users to one of two groups, but exposes them to exactly the same experience. An A/A test can be used to (i) collect data and assess its variability for power calculations and (ii) test the experimentation system (the null hypothesis should be rejected about 5 % of the time when a 95 % confidence level is used) (Kohavi et al. 2009; Martin 2008).

Planning Experiments

Several aspects of planning an experiment are important: estimating adequate sample size, gathering the right metrics, tracking the right users, and randomization unit.

Sample size. Sample size is determined by the percent of users admitted into the experiment variants (control and treatments) and how long the experiment runs. As an experiment runs longer, more visitors are admitted into the variants, so sample sizes increase. Experimenters can choose the relative percent of visitors that are in the control and treatment which affects how long you will need to run the experiment. Several authors (Deng et al. 2013; Kohavi et al. 2009) have addressed the issue of sample size and length of experiment in order to achieve adequate statistical power for an experiment, where statistical power of an experiment is the probability of detecting a given effect when it exists (technically, the probability of correctly rejecting the null hypothesis when it is false). In addition to planning an experiment for adequate power, a best practice is to run the experiment for at least one week (to capture a full weekly cycle) and then multiple weeks beyond that. When "novelty" or "primacy" effects are suspected (i.e., the initial effect of the treatment is not the same as the long-term effect), the experiment should be run long enough to estimate the asymptotic effect of the treatment. Finally, measuring the effect on high-variance metric, such as loyalty (sessions/user), will generally require more users than for other metrics (Kohavi et al. 2012).

Observations, Metrics, and the OEC. Gathering observations (i.e., logging events) so that the right metrics can be computed is critical to successful experimentation. Whenever possible and economically feasible, one should gather as many observations as possible that relate to answering potential questions of interest, whether user related or performance related (e.g., latency, utilization, crashes). We recommend computing many metrics from the observations (e.g., hundreds) because they can give rise to surprising insights, although care must be taken to correctly understand and control for the false-positive rate (Kohavi et al. 2013; Hochberg and Benjamini 1995). While having many metrics is great for insights, decisions should be made using the Overall Evaluation Criterion (OEC). See Tenet 1 earlier for a description of the OEC.

Triggering. Some treatments may be relevant to all users who come to a website. However, for many experiments, the difference introduced is relevant for a subset of visitors (e.g., a change to the checkout process, which only 10 % of visitors start). In these cases, it is best to include only those visitors who would have experienced a difference in one of the variants (this commonly requires counterfactual triggering for the control). Some architectures (Kohavi et al. 2009) trigger users into an experiment either explicitly or using lazy (or late-bound) assignment. In either case, the key is to analyze only the subset of the population that was potentially impacted. Triggering reduces the variability in the estimate of treatment effect, leading to more precise estimates. Because the diluted effect is often of interest, the effect can then be diluted (Deng and Hu 2015).

Randomization Unit. Most experiments use the visitor as the randomization unit for several reasons. First, for many changes being tested, it is important to give the user a consistent online experience. Second, most experimenters evaluate metrics at the user level, such as sessions per user and clicks per user. Ideally, the randomization by the experimenter is by a true user, but in many unauthenticated sites, a cookie stored by the user's browser is used, so in effect, the randomization unit is the cookie. In this case, the same user will appear to be different users if she comes to the site using a different browser, different device, or having deleted her cookie during the experiment. The next section will discuss how the choice of randomization unit affects how the analysis of different metrics should be carried out. The randomization unit can also affect the power of the test for some metrics.

For example, Deng et al. (2011) showed that the variance of page level metrics can be greatly reduced if randomization is done at the page level, but user metrics cannot be computed in such cases. In social-network settings, spillover effects violate the standard no-interference assumption, requiring unique approaches, such as clustering (Ugander et al. 2013).

Analysis of Experiments

If an experiment is carried out correctly, the analysis should be a straightforward application of well-known statistical methods. Of course, this is much preferred than trying to recover from a poor experimental design or implementation.

Confidence Intervals. Most reporting systems will display the treatment effect (actual and percent change) along with suitable confidence intervals. For reasonably large sample sizes, generally considered to be thousands of users in each variant, the means may be considered to have normal distributions (see Kohavi et al. (2014) for detailed guidance), making the formation of confidence intervals routine. However, care must be taken to use the Fieller theorem formula (Fieller 1954) for percent effect since there is a random quantity in the denominator.

Decision-making. A common approach to deciding if the treatment is better than the control is the usual hypothesis-testing procedure, assuming the normal distribution if the sample size is sufficient (Kohavi et al. 2009). Alternatives to this when normality cannot be assumed are transformations of the data (Bickel and Doksum 1981) and nonparametric or resampling/permutation methods to determine how unusual the observed sample is under the null hypothesis (Good 2005). When conducting a test of whether the treatment had an effect or not (e.g., a test of whether the treatment and control means are equal), a p value of the statistical test is often produced as evidence. More precisely, the p value is the probability to obtain an effect equal to or more extreme than the one observed, presuming the null hypothesis of no effect is true (Biau et al. 2010).

Another alternative is to use Bayes' theorem to calculate the posterior odds that the treatment had a positive impact versus the odds it had no impact (Stone 2013).

Analysis Units. Metrics may be defined with different analysis units, such as user, session, or other appropriate bases. For example, an ecommerce site may be interested in metrics such as revenue per user, revenue per session, or revenue per purchaser. Straightforward statistical methods (e.g., the usual t-test and variants) apply to any metric that has user as its analysis unit if users are the unit of randomization since users may be considered independent. However, if the analysis unit is not the same as the randomization unit, the analysis units may not be considered independent, and other methods need to be used to calculate standard deviation or to compare treatment to control. Bootstrapping (Efron and Tibshirani 1993) and the delta method (Casella and Berger 2001) are two commonly used methods when the analysis unit is not the same as the randomization unit.

Variance Reduction. Increasing the sample size is one way to increase power. However, online researchers are continually looking for ways to increase the power of their experiments while shortening, or at least not extending, the length of the tests. One way to do this is to use covariates such as pre-experiment user metrics, user demographics, location, equipment, software, connection speed, etc. (Deng et al. 2013) gave an example where a 50 % reduction in variance for a metric could be achieved by using only the pre-experiment metric values for the users.

Diagnostics. In order to assure the experimental results are trustworthy, every experimentation system should have some diagnostic tools built in. Graphs of the number of users in each variant, metric means, and treatment effects over time will help the researcher see unexpected problems or upsets to the experiment. In addition, diagnostic tests that trigger an alarm when an expected condition is not met should be built in. One critical diagnostic test is the "sample ratio mismatch" or SRM. A simple statistical test checks if the actual percentage for each variant is close enough to the planned percentages. We have found this one diagnostic is frequently the "canary in the coal mine" for online experiments. There are many possible ways an experiment can skew the number of visitors to one variant or another, and many of them will cause a large bias in the treatment effect. Another common useful test is that the performance, or latency, of the two versions is similar when expected to be so. It some cases, the treatment may be slower due to caching issues (e.g., cold start), or if the variant are unbalanced (e.g., 90/10 %), a shared resource like an LRU cache (Least Recently Used) will give an advantage to the larger variant (Kohavi and Longbotham 2010). When an experimentation platform allows overlapping experiments, a diagnostic to check for interactions between overlapping experiments is also helpful. Anytime an alarm or graph indicates a potential problem, the researcher should investigate to determine the source.

Robot Removal. Robots must be removed from any analysis of web data since their activity can severely bias experiment results; see Kohavi et al. (2009). Some robots may slip through robot-filtering techniques and should be considered when diagnostics suggest there may be a problem with the experiment.

Summary

The Internet and online connectivity of client software, websites, and online services provide a fertile ground for scientific testing methodology. Online experimentation is now recognized as a critical tool to determine whether a software or design change should be made. The benefit of experimenting online is the ability to set up a software platform for conducting the tests, which makes experimentation much more scalable and efficient and allows evaluating ideas quickly.

Recommended Reading

Biau DJ, Jolles BM, Porcher R (2010) P value and the theory of hypothesis testing. Clin Orthop Relat Res 468(3):885–892

Bickel PJ, Doksum KA (1981) An analysis of transformations revisited. J Am Stat Assoc 76(374):296–311. doi:10.1080/01621459.1981.10477649

Blank SG (2005) The four steps to the epiphany: successful strategies for products that win. Cafepress.com.

Box GEP, Hunter JS, Hunter WG (2005) Statistics for experimenters: design, innovation, and discovery. Wiley, Hoboken

Casella G, Berger RL (2001) Statistical inference, 2nd edn. Cengage Learning. http://www.amazon.com/Statistical-Inference-George-Casella

Deng A, Hu V (2015) Diluted treatment effect estimation for trigger analysis in online controlled experiments. In: WSDM, Shanghai 2015

Deng A, Xu Y, Kohavi R, Walker T (2013) Improving the sensitivity of online controlled experiments by utilizing pre-experiment data. In: WSDM, Rome 2013

Deng S, Longbotham R, Walker T, Xu Y (2011) Choice of randomization unit in online controlled experiment. In: Joint statistical meetings proceedings, Miami Beach, pp 4866–4877

Efron B, Tibshirani RJ (1993) An introduction to the bootstrap. Chapman & Hall, New York

Fieller EC (1954) Some problems in interval estimation. J R Stat Soc Ser B 16(2):175–185. doi:JSTOR2984043

Good PI (2005) Permutation, parametric and bootstrap tests of hypotheses, 3rd edn. Springer, New York

Goward C (2012) You should test that: conversion optimization for more leads, sales and profit or the art and science of optimized marketing. Sybex. http://www.amazon.com/You-Should-Test-That-Optimization/dp/1118301307

Hochberg Y Benjamini Y (1995) Controlling the false discovery rate: a practical and powerful approach to multiple testing Series B. J R Stat Soc 57(1):289–300

Kaushik A (2006) Experimentation and testing: a primer. Occam's razor. http://www.kaushik.net/avinash / 2006 / 05 / experimentation-and-testing-a-primer.html. Accessed 22 May 2008

Kohavi R, Deng A, Frasca B, Longbotham R, Walker T, Xu Y (2012) Trustworthy online controlled experiments: five puzzling outcomes explained. In: Proceedings of the 18th conference on knowledge discovery and data mining. http://bit.ly/expPuzzling

Kohavi R, Deng A, Frasca B, Walker T, Xu Y, Pohlmann N (2013) Online controlled experiments at large scale. In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining (KDD 2013). http://bit.ly/ExPScale

Kohavi R, Deng A, Longbotham R, Xu Y (2014) Seven rules of thumb for web site. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining (KDD '14). http://bit.ly/expRulesOfThumb

Kohavi R, Longbotham R (2010) Unexpected results in online controlled experiments. In: SIGKDD Explorations. http://bit.ly/expUnexpected

Kohavi R, Longbotham R, Walker T (2010) Online experiments: practical lessons. IEEE Comput Sept:82–85. http://bit.ly/expPracticalLessons

Kohavi R, Longbotham R, Sommerfield D, Henne RM (2009) Controlled experiments on the web: survey and practical guide. Data Min Knowl Discov 18:140–181. http://bit.ly/expSurvey

Kohavi R, Crook T, Longbotham R (2009) Online experimentation at microsoft. In: Third workshop on data mining case studies and practice prize. http://bit.ly/expMicrosoft

Malinas G, Bigelow J (2009) Simpson's paradox. Stanford Encyclopedia of Philosophy. http://plato.stanford.edu/entries/paradox-simpson/

Manzi J (2012) Uncontrolled: the surprising payoff of trial-and-error for business, politics, and society. Basic Books. https://www.amazon.com/Uncontrolled-Surprising-Trial-Error-Business-ebook/dp/B007V2VEQO

Martin RC (2008) Clean code: a handbook of Agile software craftsmanship. Prentice Hall, Upper Saddle River

McFarland C (2012a) Experiment!: website conversion rate optimization with A/B and multivariate. New Riders. http://www.amazon.com/Experiment-Website-conversion-optimization-multivariate/dp/0321834607

McFarland C (2012b) Experiment!: website conversion rate optimization with A/B and multivariate testing. New Riders. http://www.amazon.com/Experiment-Website-conversion-optimization-multivariate/dp/0321834607

McKinley D (2013) Testing to cull the living flower. http://mcfunley.com/testing-to-cull-the-living-flower

Moran M (2007) Do it wrong quickly: how the web changes the old marketing rules. IBM Press. http://www.amazon.com/Do-Wrong-Quickly-Changes-Marketing/dp/0132255960/

Moran M (2008) Multivariate testing in action: Quicken Loan's Regis Hadiaris on multivariate testing. www.biznology.com/2008/12/multivariate_testing_in_action/

Peterson ET (2004) Web analytics demystified: a marketer's guide to understanding how your web site affects your business. Celilo Group Media and CafePress. http://www.amazon.com/Web-Analytics-Demystified-Marketers-Understanding/dp/0974358428/

Ries E (2011) The lean startup: how today's entrepreneurs use continuous innovation to create radically successful businesses. Crown Business, New York

Rubin KS (2012) Essential scrum: a practical guide to the most popular Agile process. Addison-Wesley Professional, Upper Saddle River

Schrage M (2014) The innovator's hypothesis: how cheap experiments are worth more than good ideas. MIT Press. http://www.amazon.com/Innovators-Hypothesis-Cheap-Experiments-Worth/dp/0262528967

Siroker D, Koomen P (2013) A/B testing: the most powerful way to turn clicks into customers. Wiley. http://www.amazon.com/Testing-Most-Powerful-Clicks-Customers/dp/1118792416

Stone JV (2013) Bayes' rule: a tutorial introduction to Bayesian analysis. Sebtel Press. http://www.amazon.com/Bayes-Rule-Tutorial-Introduction-Bayesian/dp/0956372848

Tang D, Agarwal A, O'Brien D, Meyer M (2010) Overlapping experiment infrastructure: more, better, faster experimentation. In: KDD 2010: The 16th ACM SIGKDD international conference on knowledge discovery and data mining, Washington, DC, 25–28 July

Ugander J, Karrer B, Backstrom L, Kleinberg J (2013) Graph cluster randomization: network exposure to multiple universes. In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining (KDD '13), Chicago

Online Learning

Peter Auer
Department of Information Technology,
University of Leoben, Leoben, Austria

Abstract

Online learning and its variants are one of the main models of computational learning theory, complementing statistical PAC learning and related models. An online learner needs to make predictions about a sequence of instances, one after the other, and receives feedback after each prediction. The performance of the online learner is typically compared to the best predictor from a given class, often in terms of its excess loss (the regret) over the best predictor. Some of the fundamental online learning algorithms and their

variants are discussed: weighted majority, follow the perturbed leader, follow the regularized leader, the perceptron algorithm, the doubling trick, bandit algorithms, and the issue of adaptive versus oblivious instance sequences. A typical performance proof of an online learning algorithm is exemplified for the perceptron algorithm.

Synonyms

Mistake-bounded learning; Prediction with expert advice; Sequential learning

Definition

In the online learning model, the learner needs to make predictions or choices about a sequence of instances, one after the other, and receives a loss or reward after each prediction or choice. Typically, the learner receives a description of the current instance before making a prediction. The goal of the learner is to minimize its accumulated losses (or equivalently maximize the accumulated rewards).

The performance of the online learner is usually compared to the best predictor in hindsight from a given class of predictors. This comparison with a predictor in hindsight allows for meaningful performance bounds even without any assumptions on how the sequence of instances is generated. In particular, this sequence of instances may not be generated by a random process but by an adversary that tries to prevent learning.

In this sense performance bounds for online learning are typically worst-case bounds that hold for any sequence of instances. This is possible since the performance bounds are relative to the best predictor from a given class. Often these performance guarantees are quite strong, showing that the learner can do nearly as well as the best predictor from a large class of predictors.

Motivation and Background

Online learning is one of the main models of learning theory, complementing the statistical approach of the PAC learning model by allowing a more general process for generating learning instances. The distinctive properties of the online learning model are:

- Learning proceeds in trials,
- There is no designated learning phase, the performance of the learner is evaluated continuously from the start,
- No assumptions on the generation of the inputs to the learner are necessary; they may depend even adversarially on previous predictions of the learner,
- Sequential predictions model an interaction between the learner and its environment,
- Performance guarantees for learning algorithms are typically relative to the performance of the best predictor in hindsight from some given class.

The first explicit models of online learning were proposed by Angluin (1988) and Littlestone (1988), but related work on repeated games by Hannan (1957) dates back to 1957. Littlestone proposed online learning as a sequence of trials, where in each the learner receives some input, makes a prediction of the associated output, and receives the correct output. It was assumed that some function from a known class maps the inputs to correct outputs. The performance of the learner is measured by the number of mistakes made by a learner, before it converges to the correct predictor. Angluin's equivalence query model of learning is formulated differently but is essentially equivalent to Littlestone's model.

The restriction that some function from the class must predict all outputs correctly was then removed, e.g., Vovk (1990) and Littlestone and Warmuth (1994). In their setting the learner competes with the best predictor from the given class. As the class of predictors can be seen as a set of experts advising the learner about the correct predictions, this led to the term “prediction with expert advice." A comprehensive treatment of binary predictions with expert advice can be found in Cesa-Bianchi et al. (1997). Relations of online learning to several other fields (e.g., compression, competitive analysis, game theory, and portfolio selection) are discussed in the excellent book on sequential prediction by Cesa-Bianchi and Lugosi (2006).

Structure of Learning System

The online learning model is formalized as follows. In each trial $t = 1, 2, \ldots$, the learner

1. Receives input $x_t \in X$,
2. Chooses a prediction or output $y_t \in Y$,
3. Receives response $z_t \in Z$,
4. Incurs loss $\ell_t = \ell(y_t, z_t)$,

where $\ell : Y \times Z \mapsto \mathbb{R}$ is some loss function. The performance of a learner up to trial T is measured by its accumulated loss $L_T = \sum_{t=1}^{T} \ell_t$. For now it is assumed that inputs x_t and responses z_t are independent from the learner's predictions y_t. Such sequences of instances are called *oblivious* to the learner. *Adaptive* sequences of instances will be discussed later.

Performance bounds for online learning algorithms are typically in respect to the performance of an optimal predictor (or expert) E^* in hindsight from some class $\mathcal{E}$, $E^* \in \mathcal{E}$. A predictor E maps the past given by $(x_1, y_1, z_1), \ldots, (x_{t-1}, y_{t-1}, z_{t-1})$ and the current input x_t to a prediction y_t^E. As for the learner, the performance of a predictor is measured by its accumulated loss $L_T^E = \sum_{t=1}^{T} \ell_t^E$, where $\ell_t^E = \ell(y_t^E, z_t)$. Most bounds for the loss of online algorithms are of the form

$$L_T \leq a \min_{E \in \mathcal{E}} L_T^E + b\mathcal{C}(\mathcal{E}),$$

where the constants a and b depend on the loss function and $\mathcal{C}(\mathcal{E})$ measures the complexity of the class of predictors (e.g., the complexity $\mathcal{C}(\mathcal{E})$ could be $\log |\mathcal{E}|$ for a finite class $\mathcal{E}$.) Often it is possible to trade the constant a against the constant b such that bounds

$$L_T \leq L_T^* + o(L_T^*)$$

can be achieved, where $L_T^* = \min_{E \in \mathcal{E}} L_T^E$ is the loss of the best predictor in hindsight up to time T. These bounds are of particular interest as they show that the loss of the learning algorithm is only little larger than the loss of the best predictor. For such bounds the regret R_T of the learning algorithm,

$$R_T = L_T - L_T^*,$$

is the relevant quantity that measures the cost of not knowing the best predictor in advance.

The next section makes this general definition of online learning more concrete by presenting some important online learning algorithms.

Theory/Solution

The Weighted Majority Algorithm

The weighted majority algorithm developed by Littlestone and Warmuth (1994) is one of the fundamental online learning algorithms, with many relatives using similar ideas. We will present it for the basic scenario with a finite set of experts $\mathcal{E}$, binary predictions $y_t \in \{0, 1\}$, binary responses $z_t \in \{0, 1\}$, and the discrete loss which just counts mistakes, $\ell(y, z) = |y - z|$, such that $\ell(y, z) = 0$ if $y = z$ and $\ell(y, z) = 1$ if $y \neq z$. (We will use the terms experts and predictors interchangeably. In the literature finite sets of predictors are often called experts.)

The weighted majority algorithm maintains a weight w_t^E for each expert $E \in \mathcal{E}$ that is initialized as $w_1^E = 1$. The weights are used to combine the predictions y_t^E of the experts by a weighted majority vote: $y_t = 1$ if $\sum_E w_t^E y_t^E \geq \frac{1}{2} \sum_E w_t^E$, and $y_t = 0$ otherwise. After receiving the response z_t, the weights of experts that made incorrect predictions are reduced by multiplying with some constant $\beta < 1$, $w_{t+1}^E = \beta w_t^E$ if $y_t^E \neq z_t$, and $w_{t+1}^E = w_t^E$ if $y_t^E = z_t$. As a performance bound for the weighted majority algorithm one can achieve

$$L_T \leq 2L_T^* + 2\sqrt{2L_T^* \log |\mathcal{E}|} + 4 \log |\mathcal{E}|$$

with $L_T^* = \min_{E \in \mathcal{E}} L_T^E$ and an appropriate β. (Better constants on the square root and the logarithmic term are possible.)

While in this bound the loss of the deterministic weighted majority algorithm is twice the loss of the best expert, the randomized version of the weighted majority algorithm almost achieves the loss of the best expert. Instead of using a deterministic prediction, the randomized weighted majority algorithm tosses a coin and predicts $y_t = 1$ with probability $\sum_E w_t^E y_t^E / \sum_E w_t^E$. Below we prove the following bound on the expected loss of the randomized algorithm:

$$\mathbb{E}\left[L_T\right] \le \frac{\log(1/\beta)}{1-\beta} L_T^* + \frac{1}{1-\beta} \log|\mathcal{E}|. \quad (1)$$

Approximately optimizing for β yields $\beta = 1 - \varepsilon$, where $\varepsilon = \min\{1/2, \sqrt{2(\log|\mathcal{E}|)/L_T^*}\}$, and

$$\mathbb{E}\left[L_T\right] \le L_T^* + \sqrt{2L_T^* \log|\mathcal{E}|} + 2\log|\mathcal{E}|. \quad (2)$$

The expectation in these bounds is only in respect to the randomization of the algorithm, no probabilistic assumptions on the experts or the sequence of responses are made. These bounds hold for any set of experts and any oblivious sequence of responses that does not depend on the randomization of the algorithm. It can be even shown that the following similar bound holds with probability $1 - \delta$ (in respect to the randomization of the algorithm):

$$L_T \le L_T^* + \sqrt{T \log(|\mathcal{E}|/\delta)}. \quad (3)$$

The proof of bound (1) shows many of the ideas used in the proofs for online learning algorithms. Key ingredients are a potential function and how the changes of the potential function relate to losses incurred by the learning algorithm. For the weighted majority algorithm, a suitable potential function is the sum of the weights, $W_t = \sum_E w_t^E$. Then, since the losses are 0 or 1,

$$\begin{aligned} \frac{W_{t+1}}{W_t} &= \frac{\sum_E w_{t+1}^E}{\sum_E w_t^E} = \frac{\sum_E \beta^{\ell_t^E} w_t^E}{\sum_E w_t^E} \\ &= \frac{\sum_E [1 - (1-\beta)\ell_t^E] w_t^E}{\sum_E w_t^E} \\ &= 1 - (1-\beta) \frac{\sum_E \ell_t^E w_t^E}{\sum_E w_t^E}. \end{aligned} \quad (4)$$

Since the probability that the randomized weighted majority algorithm makes a mistake is given by $\mathbb{E}\left[\ell_t\right] = \sum_E \ell_t^E w_t^E / \sum_E w_t^E$, we get by taking logarithms that

$$\begin{aligned} \log W_{t+1} - \log W_t &= \log(1 - (1-\beta)\mathbb{E}\left[\ell_t\right]) \\ &\le -(1-\beta)\mathbb{E}\left[\ell_t\right] \end{aligned} \quad (5)$$

(since $\log(1-x) \le -x$ for $x \in (0,1)$). Summing over all trials $t = 1, \ldots, T$, we find

$$\log W_{T+1} - \log W_1 \le -(1-\beta)\mathbb{E}\left[L_t\right].$$

Since $W_1 = |\mathcal{E}|$ and $W_{T+1} = \sum_E w_{T+1}^E = \sum_E \beta^{L_T^E} \ge \beta^{L_T^*}$, rearranging the terms gives (1).

Extensions and Modifications of the Weighted Majority Algorithm

Variants and improved versions of the weighed majority algorithm have been analyzed for various learning scenarios. An excellent coverage of the material can be found in Cesa-Bianchi and Lugosi (2006). In this section we mention a few of them.

General loss functions. The analysis of the weighted majority algorithm can be generalized to any convex set of predictions Y and any set of outcomes Z, as long as the loss function $\ell(y, z)$ is bounded and convex in the first argument. Typically it is possibly to derive a learning algorithm with loss bound

$$L_T \le aL_T^* + b \log|\mathcal{E}|$$

for suitable values a and b. Of particular interest is the smallest b for which a loss bound with $a =$

1 can be achieved. Some algorithms for convex prediction sets Y will be discussed later.

Tracking the best expert and other structured experts. For a large number of experts, the loss bound of the weighted majority algorithm is still interesting since it scales only logarithmically with the number of experts. Nevertheless, the weighted majority algorithm and other online learning algorithms become computationally demanding as they need to keep track of the performance of all experts (computation time scales linearly with the number of experts). If the experts exhibit a suitable structure, then this computational burden can be avoided.

As an example we consider the problem of tracking the best expert. Let $\mathcal{E}_0$ be a small set of base experts. The learning algorithm is required to compete with the best sequence of at most S experts from $\mathcal{E}_0$: the trials are divided into S periods, and in each period another expert might predict optimally. Thus the minimal loss of a sequence of S experts is given by

$$L^*_{T,S} = \min_{0=T_0\le T_1\le T_2\le\cdots\le T_S=T} \sum_{i=1}^{S} \min_{E\in\mathcal{E}_0} \sum_{t=T_{i-1}+1}^{T_i} \ell_t^E,$$

where the trials are optimally divided into S periods $[T_{i-1}+1, T_i]$, and the best base expert is chosen for each period. Such sequences of base experts can be seen as experts themselves, but the number of such compound experts is $\binom{T-1}{S-1}|\mathcal{E}_0|^S$ and thus computationally prohibitive. Fortunately, a slightly modified weighted majority algorithm applied to the base experts achieves almost the same performance as the weighted majority algorithm applied to the compound experts. The modification of the weighted majority algorithm just lower bounds the relative weight of each base expert. This allows the relative weight of a base expert to grow large quickly if this expert predicts best in the current period. Hence, also the learning algorithm will predict almost optimally in each period. A recent and improved version of the weighted majority algorithm with many related references is given in Luo and Schapire (2015).

Other examples of structured experts include tree experts and shortest path problems (see Cesa-Bianchi and Lugosi (2006) for further references). For the shortest path problem in a graph, the learner has to compete with the single best path in hindsight, while edge costs may change at each time t. In principle the weighted majority algorithm could be employed with one expert for each path, but since the number of paths is usually exponential in the size of the graph, this might be computationally infeasible. Instead, the *follow the perturbed leader* strategy can be used as an alternative to the weighted majority algorithm.

Follow the perturbed leader. This is a simple prediction strategy that was originally proposed by Hannan (1957). In each trial t, it generates identically distributed random values ψ_t^E for every expert E, adds these random values to the losses of the experts so far, and predicts with the expert that achieves the minimum sum,

$$\hat{E}_t = \arg\min_{E\in\mathcal{E}} L_{t-1}^E + \psi_t^E,$$

$$y_t = y_t^{\hat{E}_t}.$$

For carefully chosen distributions of the ψ_t^E, this prediction strategy achieves loss bounds similar to the weighted majority like algorithms.

To apply this strategy to the shortest path problem described above, it is assumed that all paths have an equal number of edges (by possibly adding dummy edges). Then instead of generating random values ψ_t^E for each path E, a random value for each edge is generated, and the value ψ_t^E for a path is given by the sum of the random values for its edges. This allows to find the best path $\hat{E}_t$ efficiently by a shortest path calculation according to the accumulated and randomly modified edge costs.

The doubling trick. The optimal choice of β in the performance bound (1) requires knowledge about the loss of the best expert L_T^*. If such knowledge is not available, the doubling trick can be used. The idea is to start with an initial guess $\hat{L}^*$ and choose β according to this guess. When the loss of the best expert exceeds this guess, the guess is doubled, β is modified, and the learning algorithm is restarted. The bound (2) increases only slightly when L_T^* is not known and the doubling trick is used. It can be shown that still

$$\mathbb{E}[L_T] \leq L_T^* + c_1\sqrt{L_T^* \log|\mathcal{E}|} + c_2 \log|\mathcal{E}|$$

for suitable constants c_1 and c_2. A thorough analysis of the doubling trick can be found in Cesa-Bianchi et al. (1997). Variations of the doubling trick can be used for many online learning algorithms to "guess" unknown quantities. A drawback of the doubling trick is that it restarts the learning algorithm and forgets about all previous trials. An alternative approach is an iterative adaptation of the parameter β, which can be shown to give better bounds than the doubling trick. The advantage of the doubling trick is that its analysis is quite simple.

Prediction with limited feedback and the multiarmed bandit problem. In the setting considered so far, the learner has full information of the past, as all past outcomes $z_1, \ldots, z_{t-1} \in \{0, 1\}$ and all predictions of the experts $y_1^E, \ldots, y_t^E$, $E \in \mathcal{E}$, are available, before prediction y_t is made. In some learning scenarios, the learner might not have such full information. One example is the multiarmed bandit problem, and a more general case is *prediction with partial monitoring*.

In the multiarmed bandit problem the learner chooses to follow one of K experts, observes the loss of this expert, and also incurs the loss of this expert. Formally, the learner chooses an expert $y_t = E_t \in \mathcal{E} = \{1, \ldots, K\}$, receives the loss of the chosen prediction $z_t = \ell_t(E_t)$, and incurs loss $\ell(y_t, z_t) = z_t = \ell_t(E_t)$. (Here $\ell_t(E)$ denotes the loss of expert E at time t.) The losses of the other experts, $\ell_t(E)$, $E \neq E_t$, are not revealed to the learner. The goal of the learner is to compete with the loss of the single best expert, $L_T^* = \min_{E \in \mathcal{E}} L_T^E$, $L_T^E = \sum_{t=1}^T \ell_t(E)$. The multiarmed bandit problem looks very much like the original online learning problem with the predictions $y \in Y$ as experts.

Since at each time t the learner observes only the loss of the chosen expert, it needs to estimate the unseen losses of the other experts and use these estimates when choosing an expert. Since accurate estimates need a sufficient amount of data, this leads to a trade-off between choosing the (apparently) best expert to minimize losses and choosing a different expert for which more data are needed. This exploration-exploitation trade-off also appears elsewhere in online learning, but it is most clearly displayed in the bandit problem. An algorithm that deals well with this trade-off is again a simple variant of the weighted majority algorithm. This algorithm does exploration trials with some small probability, and in such exploration trials, it chooses an expert uniformly at random. This algorithm has been analyzed in Auer et al. (2002) for gains instead of losses. For losses $\ell \in [-1, 0]$ the accumulated loss of the algorithm can be bounded as

$$\mathbb{E}[L_T] \leq L_T^* + 3\sqrt{K|L_T^*| \log K}.$$

Compared with (2), the regret increases only by a factor of $\sqrt{K}$. Further results, including lower bounds and results for stochastic bandit problems, are summarized in Bubeck and Cesa-Bianchi (2012). For the stochastic multiarmed bandit problem, it is assumed that the losses of the experts are generated independently at random by some distribution for each expert. This allows for specialized algorithms with substantially improved regret bounds.

A generalization of bandit problems are partial monitoring games (Bartók 2014), where the learner receives only indirect feedback about the losses of the experts. Depending on how much the feedback reveals about the incurred losses, partial monitoring games can be classified as games with either 0, $\Theta(T^{1/2})$, $\Theta(T^{2/3})$, or $\Theta(T)$ regret.

The Perceptron Algorithm

In this section we consider an example for an online learning algorithm that competes with a *continuous* set of experts, in contrast to the *finite* sets of experts we have considered so far. This algorithm—the perceptron algorithm (Rosenblatt 1958)—was among the first online learning algorithms developed. Another of this early online learning algorithms with a continuous set of experts is the Winnow algorithm by Littlestone (1988). A unified analysis of these algorithms can be found in Cesa-Bianchi and Lugosi (2006). This analysis covers a large class of algorithms, in particular the p-norm perceptrons, which smoothly interpolate between the perceptron algorithm and Winnow.

The perceptron algorithm aims at learning a linear classification function. Thus inputs are from a Euclidean space, $X = \mathbb{R}^d$, the predictions and responses are binary, $Y = Z = \{0, 1\}$, and the discrete misclassification loss is used. Each expert is a linear classifier, represented by its weight vector $v \in \mathbb{R}^d$, whose linear classification is given by $\Phi_v : X \to \{0, 1\}$, $\Phi_v(x) = 1$ if $v \cdot x \geq 0$ and $\Phi_{v,\theta}(x) = 0$ if $v \cdot x < 0$.

The perceptron algorithm maintains a weight vector $w_t \in \mathbb{R}^d$ that is initialized as $w_1 = (0, \ldots, 0)$. After receiving input x_t, the perceptron's prediction is calculated using this weight,

$$y_t = \Phi_{w_t}(x_t),$$

and the weight vector is updated,

$$w_{t+1} = w_t + \eta(z_t - y_t)x_t,$$

where $\eta > 0$ is a learning rate parameter. Thus, if the prediction is correct, $y_t = z_t$, then the weights are not changed. Otherwise, the product $w_{t+1} \cdot x_t$ is moved into the correct direction: since $w_{t+1} \cdot x_t = w_t \cdot x_t + \eta(z_t - y_t)||x_t||^2$, $w_{t+1} \cdot x_t > w_t \cdot x_t$ if $y_t = 0$ but $z_t = 1$, and $w_{t+1} \cdot x_t < w_t \cdot x_t$ if $y_t = 1$ but $z_t = 0$.

We may assume that the inputs are normalized, $||x_t|| = 1$, otherwise a normalized x_t can be used in the update of the weight vector. Furthermore, we note that the learning rate η is irrelevant for the performance of the perceptron algorithm, since it scales only the size of the weights but does not change the predictions. Nevertheless, we keep the learning rate since it will simplify the analysis.

Analysis of the perceptron algorithm. To compare the perceptron algorithm with a fixed (and optimal) linear classifier v, we again use a potential function, $||w_t - v||^2$. For the change of the potential function when $y_t \neq z_t$, we find

$$\begin{aligned} &||w_{t+1} - v||^2 - ||w_t - v||^2 \\ &= ||w_t + \eta(z_t - y_t)x_t - v||^2 - ||w_t - v||^2 \\ &= ||w_t - v||^2 + 2\eta(z_t - y_t)(w_t - v) \cdot x_t \\ &+ \eta^2(z_t - y_t)^2\, ||x_t||^2 - ||w_t - v||^2 \\ &= 2\eta(z_t - y_t)(w_t \cdot x_t - v \cdot x_t) + \eta^2. \end{aligned}$$

Since $w_t \cdot x_t < 0$ if $y_t = 0$ and $w_t \cdot x_t \geq 0$ if $y_t = 1$, we get $(z_t - y_t)(w_t \cdot x_t) \leq 0$ and

$$||w_{t+1} - v||^2 - ||w_t - v||^2 \leq -2\eta(z_t - y_t)(v \cdot x_t) + \eta^2.$$

Analogously, the linear classifier v makes a mistake in trial t if $(z_t - y_t)(v \cdot x_t) < 0$, and in this case $-(z_t - y_t)(v \cdot x_t) \leq ||v||$. Hence, summing over all trials (where $y_t \neq z_t$) gives

$$\begin{aligned} &||w_{T+1} - v||^2 - ||w_1 - v||^2 \leq -2\eta \\ &\sum_{t:\ell_t = 1, \ell_t^v = 0} |v \cdot x_t| + 2\eta||v||L_T^v + \eta^2 L_T, \qquad (6) \end{aligned}$$

where the sum is over all trials where the perceptron algorithm makes a mistake but the linear classifier v makes no mistake. To proceed, we assume that for the correct classifications of the linear classifier v, the product $v \cdot x_t$ is bounded away from 0 (which describes the decision boundary). We assume $|v \cdot x_t| \geq \gamma_v > 0$. Then

$$\begin{aligned} ||w_{T+1} - v||^2 - ||w_1 - v||^2 &\leq -2\eta\gamma_v(L_T - L_T^v) \\ &+ 2\eta||v||L_T^v + \eta^2 L_T, \qquad (7) \end{aligned}$$

and

$$L_T(2\eta\gamma_v - \eta^2) \leq ||v||^2 + L_T^v(2\eta\gamma_v + 2\eta||v||),$$

since $||w_{T+1} - v||^2 \geq 0$ and $w_1 = (0, \ldots, 0)$. For $\eta = \gamma_v$ the following loss bound for the perceptron algorithm is achieved:

$$L_T \leq ||v||^2/\gamma_v^2 + 2L_T^v(1 + ||v||/\gamma_v).$$

Thus the loss of the perceptron algorithm does not only depend on the loss of an (optimal) linear classifier v but also on the gap by which the classifier can separate the inputs with $z_t = 0$ from the inputs with $z_t = 1$. The size of this gap is essentially given by $\gamma_v/||v||$.

Relation between the perceptron algorithm and support vector machines. The gap $\gamma_v/||v||$ is the quantity maximized by support vector machines, and it is the main factor determining the prediction accuracy (in a probabilistic sense) of a support vector machine. It is not coincidental that the same quantity appears in the performance bound of the perceptron algorithm, since it measures the difficulty of the classification problem.

As for support vector machines, kernels $K(\cdot,\cdot)$ can be used in the perceptron algorithm. For that, the dot product $w_t \cdot x_t$ is replaced by the kernel representation $\sum_{\tau=1}^{t-1}(z_\tau - y_\tau)K(x_\tau, x)$. Obviously this has the disadvantage that all previous inputs for which mistakes were made must be kept available.

Online Convex Optimization

For online convex optimization, the learner has to choose a prediction y_t from some convex set Y, receives as feedback a convex loss function $L_t : Y \to \mathbb{R}$, and suffers loss $\ell_t = L_t(y_t)$. An excellent exposition of online convex optimization is given in Shalev-Shwartz (2011).

Many online learning problems and algorithms can be cast in the framework of online convex optimization, in particular also the weighted majority algorithm and the perceptron algorithm. While both algorithms make binary predictions $y_t \in \{0, 1\}$ and suffer a discrete loss, they both can be *convexified*: For the weighted majority algorithm we can consider the probability p_t for $y_t = 1$ as the prediction of the algorithm, such that the expected loss is $\mathbb{E}\ell(y_t, z_t) = |z_t - p_t|$ which is a convex function in p_t. For the perceptron algorithm, the discrete loss can be upper bounded by a convex *surrogate loss function*, and the loss analysis can be done in respect to these surrogate loss functions.

Two simple but often effective strategies for online convex optimization are *follow the leader* and *follow the regularized leader*. The *follow the leader* strategy chooses the prediction that would minimize the accumulated loss so far,

$$y_t = \arg\min_{y \in Y} \sum_{i=1}^{t-1} L_i(y).$$

For some online convex optimization problems, this gives very good regret bounds, in particular for online quadratic optimization, where $Y = \mathbb{R}^d$ and $L_t(y) = ||y - z_t||^2$ is the squared Euclidean distance to some $z_t \in \mathbb{R}^d$. It can be shown that in this case, the regret is bounded by

$$\min_y \sum_{t=1}^T ||y - z_t||^2 - \sum_{t=1}^T ||y_t - z_t||^2 \leq 4Z^2(1 + \log T), \quad (8)$$

where $Z = \max_{1 \leq t \leq T} ||z_t||$. This strategy fails, though, for other loss functions, for example, for online linear optimization with losses $L_t(y) = y \cdot z_t$, when the regret might be as large as $\Omega(T)$. This problem can be avoided by the *follow the regularized leader* strategy which chooses predictions

$$y_t = \arg\min_{y \in Y} \left[\sum_{i=1}^{t-1} L_i(y) + R(y) \right]$$

for some regularization function $R : Y \to \mathbb{R}$. For online linear optimization with quadratic regularization $R(y) = \frac{Z}{B}\sqrt{\frac{T}{2}}||y||^2$, *follow the regularized leader* achieves regret

$$\min_{y:||y|| \leq B} \sum_{t=1}^T (y \cdot z_t) - \sum_{t=1}^T (y_t \cdot z_t) \leq ZB\sqrt{2T},$$

if all $||z_t|| \leq Z$. Online convex optimization is a very active field of research, and a good starting point is the exposition (Shalev-Shwartz 2011).

Oblivious Versus Adaptive Instance Sequences

So far we have assumed that the sequence of instances is not influenced by the predictions of the learner. If the sequence of instances is adaptive and depends on the predictions of the learner, additional care is necessary. In particular the definition of regret is subtle: since the instances depend on the predictions, the instances encountered by the learner may be very different from the instances encountered when following the prediction of a single expert. Therefore, it is in general not possible to bound the loss of a learner by the losses the experts would have incurred when making their predictions. This notion of regret is called *policy regret* (Dekel 2012), and it is easy to construct examples where any learning algorithm suffers $\Omega(T)$ loss while the predictions of the best expert suffer zero loss. To obtain nontrivial bounds on the policy regret, the adaptiveness of the instance sequence needs to be restricted, for example, by a bounded memory assumption: the instance at time t may depend only on the last m predictions of the learner.

In contrast, the loss of the learner can often be bounded by the loss of the best expert *on the sequence generated in response to the predictions of the learner.* The difference between the loss of the learner and the loss of the best expert on the same sequence of instances is called the *external regret*. As explained above, the notion of external regret is not fully satisfactory for adaptive sequences, but it allows to carry over many result for oblivious sequences to adaptive sequences. For an example, the high probability bound (3) for the weighted majority algorithm,

$$L_T \leq L_T^* + \sqrt{T \log(|\mathcal{E}|/\delta)},$$

holds also for any adaptive instance sequence that depends on the past predictions of the *learner*.

Recommended Reading

Angluin D (1988) Queries and concept learning. Mach Learn 2:319–342

Auer P, Cesa-Bianchi N, Freund Y, Schapire R (2002) The nonstochastic multiarmed bandit problem. SIAM J Comput 32:48–77

Bartók G, Foster D, Pál D, Rakhlin A, Szepesvári C (2014) Partial monitoring—classification, regret bounds, and algorithms. Math Oper Res 39: 967–997

Bubeck S, Cesa-Bianchi N (2012) Regret analysis of stochastic and nonstochastic multi-armed bandit problems. Found Trends Mach Learn 5:1–122

Cesa-Bianchi N, Freund Y, Haussler D, Helmbold D, Schapire R, Warmuth M (1997) How to use expert advice. JACM 44:427–485

Cesa-Bianchi N, Lugosi G (2006) Prediction, learning, and games. Cambridge University Press, Cambridge/New York

Dekel O, Tewari A, Arora R (2012) Online bandit learning against an adaptive adversary: from regret to policy regret. In: Proceedings of the 29th international conference on machine learning, Edinburgh

Hannan J (1957) Approximation to Bayes risk in repeated play. Contrib Theory Games 3:97–139

Littlestone N (1988) Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. Mach Learn 2:285–318

Littlestone N, Warmuth M (1994) The weighted majority algorithm. Inf Comput 108:212–261

Luo H, Schapire RE (2015) Achieving all with no parameters: Adanormalhedge. In: Proceedings of the 28th conference on learning theory, Paris, pp 1286–1304

Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65:386–408

Shalev-Shwartz S (2011) Online learning and online convex optimization. Found Trends Mach Learn 4: 107–194

Vovk V (1990) Aggregating strategies. In: Proceedings of 3rd annual workshop on computational learning theory, Rochester. Morgan Kaufmann, pp 371–386

Ontology Learning

Different approaches have been used for building ontologies, most of them to date mainly using manual methods (▶ Text Mining for the Semantic Web). An approach to building ontologies was set up in the CYC project, where the main step involved manual extraction of common sense knowledge from different sources. Ontology construction methodologies usually involve several phases including *identifying the purpose of the ontology* (why to build it, how will it be used, the range of the users), *building the ontology*, *evalu-*

ation and documentation. Ontology learning relates to the phase of building the ontology using semiautomatic methods based on text mining or machine learning.

Opinion Extraction

► Sentiment Analysis and Opinion Mining

Opinion Mining

► Sentiment Analysis and Opinion Mining

Opinion Stream Mining

Myra Spiliopoulou[1], Eirini Ntoutsi[2,3], and Max Zimmermann[4]
[1]Otto-von-Guericke University-Magdeburg, Magdeburg, Germany
[2]Leibniz Universität Hannover, Hannover, Germany
[3]Ludwig Maximilians Universität München, Munich, Germany
[4]Swedish Institute of Computer Science (SICS Swedish ICT), Kista, Sweden

Abstract

Opinion stream mining aims at learning and adaptation of a polarity model over a stream of opinionated documents, i.e., documents associated with a polarity. They comprise a valuable tool to analyze the huge amounts of opinions generated nowadays through the social media and the Web. In this chapter, we overview methods for polarity learning in a stream environment focusing especially on how these methods deal with the challenges imposed by the stream nature of the data, namely the nonstationary data distribution and the single pass constraint.

Synonyms

Mining a Stream of Opinionated Documents; Polarity Learning on a Stream

Definition

Opinion stream mining is a variant of stream mining, of text mining and of opinion mining. Its goal is learning and adaptation of a polarity model over a stream of opinionated documents. An "opinionated document" is a text associated with a "polarity." Polarity is a value that represents the "strength" and the "direction" of an opinion. The strength can be a categorical value (e.g., $+$, $-$) or a ranking value (e.g., zero to five stars) or a continuous value (e.g., in the interval $[0, 1]$). The direction refers to whether the opinion is positive, negative, or neutral. Strength and direction are often mixed. For example, in a ranking using stars, five stars may stand for a very positive opinion, zero stars for a very negative one, and three stars for a neutral one.

As a variant of stream mining, opinion stream mining is subject to challenges of learning on a stream: adapting to changes in the data generating distribution – a phenomenon often called *concept drift* and processing the data as they arrive (in a single pass), since they cannot be retained permanently.

As a variant of text mining, opinion stream mining is subject to challenges of learning from texts: identifying the parts of speech that are in the text (e.g., verbs, adjectives, etc.); bringing the individual words into stem form (e.g., "opinions"→"opinion"); deciding which words will constitute the feature space and which are not informative and should be ignored; modeling the similarity between texts, taking (among other issues) differences in the length of texts into account; extracting the "entities" from

Work partially done while with the Ludwig-Maximilians University, Munich.

the text (e.g., persons, products); and detecting the "topics" of discourse in the texts.

As a variant of opinion mining, opinion stream mining faces further challenges: distinguishing between words that bear sentiment (e.g., "nice," "ugly") and those referring to facts (e.g., "sauna," "phone") and discerning different forms of sentiment (e.g., anger, joy). For static data, these challenges are addressed with techniques of natural language processing (NLP), text mining, and **Sentiment Analysis and Opinion Mining** (cf. lemma).

The aforementioned challenges are exacerbated in the stream context. Opinion stream mining provides solutions for learning and adapting a polarity model in a volatile setting: the topics in the opinionated documents may change; the attitude of people toward an entity (e.g., person, product, event) may change; the words used by people to express polarity may change; and even the words used by people, i.e., the vocabulary, may also evolve over time.

Motivation, Main Tasks, and Challenges

With the rise of WEB 2.0, more and more people use social media to upload opinions on essentially every subject – on products, persons, institutions, events, and topics of discourse. These accumulating opinionated data are valuable sources of information that can deliver valuable insights on the popularity of events; on the properties of products that are deemed important; on the positive or negative perception people have toward a product, person, or institution; on their attitude toward a specific subject of discourse; etc.

Background: The analysis of opinionated data is investigated in the research areas of *sentiment analysis* and *opinion mining*. These two areas overlap, whereby research on sentiment analysis puts more emphasis in understanding different types of "sentiment" (e.g., irony, anger, sadness, etc.), while opinion mining focuses more on learning models and discerning trends from data that simply have positive or negative "polarity" (or are neutral). For an extensive discussion of the subject, the reader is referred to the lemma **Sentiment Analysis and Opinion Mining**.

In Liu (2012), Bing Liu defines four opinion mining tasks as follows:

1. *Entity extraction:* "Extract all entity expressions in a document, and categorize or group synonymous entity expressions into entity clusters. Each entity expression cluster indicates a unique entity e_i."
2. *Property extraction:* "Extract all property expressions of the entities, and categorize these property expressions into clusters. Each property expression cluster of entity e_i represents a unique property a_{ij}."
3. *Opinion holder extraction:* "Extract opinion holders for opinions from text or structured data and categorize them. The task is analogous to the above two tasks."
4. *Sentiment classification:* "Determine whether an opinion on a property a_{ij} is positive, negative, or neutral, or assign a numeric sentiment rating to the property."

Among these tasks, the first one is not peculiar to opinion mining: *entity extraction* (EEX) is a subtask of document analysis. A widespread special case of EEX is named-entity recognition (NER); a minister is an entity, and a specific minister is a named entity. The goal of EEX and NER is to identify and annotate all entities in a document. To this purpose, NLP techniques are used, as well as collections of "named entities"; a list of the towns in a country is an example of such a collection.

The second task can be generalized in two ways. First, the properties need not be associated to an explicitly defined entity (e.g., a person or city); they may also be topics or subtopics under a subject of discourse (e.g., air pollution as a subtopic of environment pollution). Further, clustering is not the only way of identifying properties/topics: aspect-based opinion mining is a subdomain of topic modeling (cf. lemma **Topic Models for NLP Applications** for the general domain). In this subdomain, a document is perceived as a mixture of topics and sentiments.

In opinion *stream* mining, the collection of opinionated documents is not perceived as a static set but as an ongoing stream. While the first and third of the aforementioned tasks remain largely unchanged, the second and forth task must be redefined in the stream context. The task of property extraction on the stream is addressed with methods of *dynamic topic modeling* (see Blei and Lafferty (2006) for the core concepts) and with methods of text stream clustering (Aggarwal and Yu 2006).

The task of sentiment classification becomes a stream classification problem for an evolving text stream. Hereafter, we denote this task as "learning a polarity model" or simpler "polarity model learning," without referring explicitly to the fact that the model is learned on a stream.

Challenges of opinion stream mining: The challenges faced in opinion stream mining for property extraction and polarity learning emanate from the different aspects of volatility in the opinionated stream:

(a) *The data evolve with respect to the target variable:* The attitude of people toward a subject of discourse, a person, a product, or some property of this product may change over time. This corresponds to a change in the priors of the polarity class.
(b) *The topics evolve:* New subjects of discourse emerge, some product properties become uninteresting while others gain momentum. The learning algorithm must recognize that people discuss different topics.
(c) *The vocabulary evolves:* New words show up, some words fall out of use, and the polarity of some words may change. This means that the high-dimensional feature space used by the learning algorithm changes during the process of learning and adaption.
(d) *Labels are scarce:* In conventional stream classification, it is assumed that fresh labels are timely available for classifier adaption. Opinionated streams are fast and the inspection of opinions is a tedious task. So, the demand for human intervention/supervision for document labeling must be limited.

Main tasks of opinion stream mining: In response to challenges (a) and (c), opinion stream mining encompasses solutions for polarity model learning and adaption and also when the class priors change and when the vocabulary evolves. Next to fully supervised solutions, there are also semi-supervised learning methods and active learning methods, in response to challenge (d). In the following, we elaborate on supervised, semi-supervised, and active stream mining approaches for the classification of opinionated streams.

For challenge (b), we refer the reader to literature on text stream clustering, starting, e.g., with Aggarwal and Yu (2006), and to literature on dynamic topic modeling, starting with Blei and Lafferty (2006) and Wang and McCallum (2006). Dynamic topic modeling for opinionated document streams gained momentum in the last years, resulting in several works on dynamic topic mixture models that capture both aspects (properties) and sentiment. An example is Fu et al. (2015) on dynamic nonparametric hierarchical Dirichlet process topic modeling. An important characteristic of this work is that the number of topics can be determined automatically and adjusted over time. Further, an aging (time-decay) component is incorporated into the learning process; this allows for forgetting old topics (Fu et al. 2015). As we discuss in the next section, the issue of forgetting is also essential in supervised learning over the stream, as means of adaptation to concept drift.

Polarity Learning in an Opinionated Stream

Polarity learning is a supervised task that involves model learning and model adaption over an opinionated stream, i.e., an infinite sequence D of arriving *instances* $d_1, \cdots, d_i, \cdots$. An instance/opinionated document is a vector over a *word vocabulary* V, which is built up and changes over time.

An instance has a polarity label c. We denote the class attribute by C. Much of the research on opinion stream mining considers streams where documents have positive or negative polarity and are mixed with neutral documents. We use this convention in the following, i.e., we assume that the polarity label is one of positive (+), negative (−), or neutral (∅).

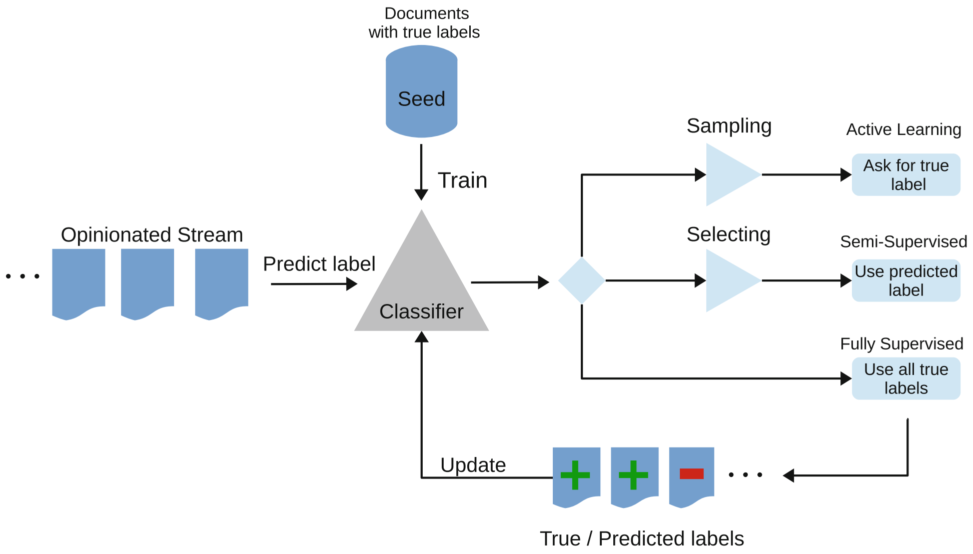
Workflow

The fully supervised stream learning scenario implies that the model is continuously learned on arriving, labeled instances. To deal with the label scarcity challenge, opinion stream mining research also contributes semi-supervised methods that learn with only an initial seed of labeled instances and active learning methods that request a label for only some of the arriving instances. An abstract workflow of the learning tasks is depicted in Fig. 1, distinguishing among supervised, semi-supervised, and active learning.

As can be seen in the figure, an initial classifier is trained on a starting set of manually labeled instances *Seed*. This set can be a small corpus of carefully selected opinionated documents that are representative of the stream, at least at the beginning, or the *Seed* can consist solely of the first arriving documents in the stream. Labels delivered by a human expert are denoted in the figure as "true labels," as opposed to the "predicted labels" that are assessed by the classifier.

In each subsequent step, the classifier predicts the labels of the arriving documents. For supervised learning, a human expert immediately delivers the true labels, which are then used for model adaption. In semi-supervised learning, the classifier adapts by using (a selection of) instances with predicted labels. In active learning, the expert is asked to deliver true labels only for some of the arriving documents which are then used for model adaption. These three ways of polarity learning are discussed hereafter.

The instances of the stream may be processed one by one as they arrive, or they may be stored into "chunks" (also called "blocks" or "batches"). In the first case, i.e., in "instance-based" processing, the classifier is adapted after seeing each new instance. In "chunk-based" processing, the classifier adapts after each chunk. A chunk may be a fixed-sized



Opinion Stream Mining, Fig. 1 Polarity learning on a stream of opinionated documents – fully supervised, semi-supervised, and active learning options block of documents or it may be defined at different levels of temporal granularity, e.g., hourly, daily, or weekly. Instance-based processing allows for fast adaption; however, the processing cost is higher as the model is updated after each instance. Chunk-based processing is more appropriate for streams where changes in the topics and/or vocabulary are manifested gradually. A detailed discussion of instance- vs chunk-based methods can be found in the lemma **Stream Classification**.

Fully Supervised Opinion Stream Classification

Fully supervised polarity learning on an opinionated stream is performed in the same way as stream classification in a conventional stream. The reader is referred to the lemma **Stream Classification** for a detailed elaboration on the interaction between the classifier and the stream, the detection of drift, and the adaption of the model. For opinionated streams, two aspects are of particular interest: how to choose a classification algorithm for polarity learning and how to deal with changes in the vocabulary.

Stream classification algorithms for polarity learning. Since there are many stream classification algorithms, it is reasonable to investigate how appropriate they are for learning on an opinionated stream. Several comparative studies have emerged at the beginning of the decade, including Bifet and Frank (2010) and Gokulakrishnan et al. (2012). In Gokulakrishnan et al. (2012), Gokulakrishnan et al. study a Twitter stream (i.e., a stream of short texts) and evaluate multinomial Naive Bayes (MNB), support vector machines (SVM), Bayesian logistic regression, sequential minimal optimization (SMO), and random forests (RF); they show that Bayesian classifiers, RF, and SMO outperform the other methods. In Bifet and Frank (2010), Bifet et al. compare MNB, stochastic gradient descend (SGD), and a Hoeffding tree (HT) algorithm; they report that MNB and SGD perform comparably when the stream is stable, but MNB has difficulties in adapting to drifts. In terms of efficiency, MNB is the fastest and HT is the slowest.

In their survey on concept drift adaption (Gama et al. 2014), Gama et al. elaborate on how *forgetting* of old data can be used to adjust a model to drift, and they discuss different forgetting strategies. The Hoeffding tree variant AdaHT (Bifet and Gavaldà 2009) forgets subtrees if performance degrades. In an opinionated stream, it is reasonable to also forget *words*, i.e., parts of the feature space, since the choice of words used in the data (here: documents!) may also change. The MNB variant proposed in Wagner et al. (2015) quantifies the contribution of a word to the polarity model by considering the number of documents containing this word and the recency of these documents; this variant is shown to adapt well to changes in the stream.

Stream classification algorithms for an evolving vocabulary. The problem of vocabulary evolution is rarely investigated in the context of stream mining. There are studies on online topic modeling and clustering on text streams, in which the model is adapted when the vocabulary – the feature space – changes (AlSumait et al. 2008; Gohr et al. 2009; Zimmermann et al. 2016), but most studies assume that all words are known in advance, and only their contribution to the model may change over time.

Among the stream classification algorithms, adaption to an evolving vocabulary is possible for some algorithms. The Hoeffding tree variant AdaHT (Bifet and Gavaldà 2009) can forget deprecated words when it forgets parts of the model (subtrees) and may be able to include new words when it builds new subtrees. The multinomial Naive Bayes variant proposed in Wagner et al. (2015) does modify the vocabulary, by considering at each timepoint only words that appear often in recent documents.

Adaption to an evolving vocabulary is an open problem. Currently, only few stream classification algorithms can deal with changes in the feature space. How to employ other classification algorithms over the opinionated stream? The fallback solution is to extend the workflow by a task that regularly recomputes the vocabulary/feature space from the most recent documents and then re-initializes the polarity model. This solution has the disadvantage that the old model is completely forgotten, but the advantage that any stream classification algorithm can be used for learning.

Semi-supervised Opinion Stream Classification

Goal of semi-supervised stream learning is to learn a model on an initial set of manually labeled documents, sometimes called the "seed set" or "initial seed," and then adapt the model by using the arriving unlabeled instances. Semi-supervised methods have the inherent advantage of not demanding human intervention after the initialization of the model.

For this family of methods, the initial seed is the only available ground truth. Hence, it is essential that the instances comprising the seed set are a representative sample. Evidently, this sample ceases being representative, as soon as concept drift occurs. Semi-supervised learning algorithms adapt to drift by building a training set that consists of the initial seed and arriving unlabeled instances, to which they themselves assign the labels. There are two strategies for the selection of unlabeled instances to be labeled by the classifier and added to the training set. The first strategy chooses instances on the grounds of the classifier's confidence to the predicted labels. The second strategy chooses instances by considering their similarity to previously labeled instances.

First strategy. Chapelle et al. point out that "Probably the earliest idea about using unlabeled data in classification is self-learning, which is also known as self-training, self-labeling, or decision-directed learning. This is a wrapper-algorithm that repeatedly uses a supervised learning method. It starts by training on the labeled data only. In each step a part of the unlabeled points is labeled according to the current decision function; then the supervised method is retrained using its own predictions as additional labeled points . . . " (Chapelle et al. 2006). However, self-training may lead to performance deterioration, because erroneous predictions of the classifier lead to erroneous labels in the training set.

Another approach is the "co-training" of several independent classifiers (Blum and Mitchell 1998). In the context of text classification, Aggarwal and Zhai propose to split the feature space into subsets and train an independent classifier on each subset (Aggarwal and Zhai 2014); then, high-confidence predictions of each single classifier are used to feed the other classifiers with new labels, so that no classifier is trained on its own predictions.

An example of co-training on a stream of tweets is in Liu et al. (2013): the complete feature space encompasses both text features (such as adjectives) and non-text features (e.g., emoticons). Views are built over this feature space, and a classifier (multiclass SVM) is trained on each view, using a small set of labeled instances only.

Second strategy. As an alternative to self-training and co-training, the second semi-supervised strategy adds to the training set those instances that are most similar to already labeled instances. One way to capitalize on labeled instances under this strategy is to cluster labeled and unlabeled instances together, then determine the label of each cluster from the labeled instances in it, and finally select for training some unlabeled instances per cluster (e.g., those closest to the cluster center).

In the context of opinionated semi-supervised stream learning, a clustering-based strategy brings two advantages. First, text stream clustering algorithms can be used, whereupon the clusters are updated gradually, as new unlabeled instances arrive. Further, these clusters reflect the properties/topics in the opinionated stream, thus addressing challenge (b) of task 2 on opinionated streams (cf. section on "Motivation, Main Tasks, and Challenges"). Example methods have been proposed by Gan et al. (2013) and by Zimmermann et al. (2015a).

In the previous section on fully supervised learning, we point out that forgetting (old data, part of the model, part of the feature space) may be beneficial for model adaption (cf. Gama et al. 2014). When learning in a semi-supervised way, though, forgetting may have negative side effects: since the seed set is the only ground truth provided by the human expert, forgetting those “precious” data labels is likely to lead to performance deterioration.

Active Learning for Opinion Stream Classification

Similarly to semi-supervised approaches, active learning methods attempt to learn and adapt to the ongoing stream without demanding a label for each arriving instance. Instead of re-acting to the labels that become available, active methods *proactively* (thereof the name “active”) request labels for the instances expected to be most informative for learning.

In active stream learning, there are two ways of requesting labels for some of the arriving instances. In the pool-based scenario, unlabeled instances are collected into a pool; the active learning algorithm chooses a subset of them and asks for their labels. In the sequential scenario, the algorithm decides for each arriving instance whether it will request a label for it. An overview of active learning methods for conventional streams is in Zliobaite et al. (2011).

Active learning is often used for various text mining tasks, including sentiment classification (Zhou et al. 2013). Active algorithms for opinionated streams also gain momentum. CloudFlows is a cloud-based platform for opinion stream mining that adheres to the pool-based scenario (Saveski and Grcar 2011; Kranjc et al. 2015): a first model of the stream is learned from a large corpus of tweets that contain emoticons; after initialization, the stream is partitioned into chunks, and an active learning algorithm is used to select instances and store them in a pool. The instances in the pool are ranked, and the top-ranked positions are shown to human experts. This approach has the advantage that human experts (e.g., in crowdsourcing) label the opinionated documents shown to them offline, whereupon these newly labeled instances are used for classifier adaption.

The algorithm ACOSTREAM (Zimmermann et al. 2015b) adheres to the sequential scenario, in the sense that sampling is done for each instance individually at its arriving time. This algorithm uses a variant of multinomial naive Bayes for classification, which (as in Wagner et al. 2015) deals with changes in the vocabulary of the arriving documents.

The multiclass active learning algorithm of Cheng et al. (2013) combines uncertainty and likelihood sampling to choose instances that are close to the current decision boundary, as well as instances from a yet unseen part of the data space. This algorithm (which adheres to the sequential scenario) is particularly interesting for learning on text streams, where some of the most recent instances may belong to an area of the data space that did not contain any instances in the past.

Recent Developments

Opinion stream mining builds upon advances in opinion mining, stream classification, active stream learning, and semi-supervised stream learning. Traditional methods in this domain have not been designed with big data in mind. However, opinionated streams have big data characteristics: volume, variability, variety, and veracity.

Volume refers to the huge number of opinions uploaded daily in social media and to the high dimensionality of the opinionated documents.

Variability refers to changes in the data flow rate and to changes in the data distribution, i.e., to concept drift.

Variety refers to the heterogeneous data types, including plain texts, images, and videos. The graph structure of the social networks, in which opinion holders are linked to each other, also adds to the variety of the data relevant to opinion mining.

Veracity refers to the uncertainty of the polarity labels provided by the human experts: labeling an opinionated stream is an inherently difficult task, since some opinionated documents (e.g., documents containing subtle irony) may be perceived differently by different people.

Challenges associated to these four Vs are not always peculiar to opinion stream mining: while challenges associated to variability are exacerbated in the opinion stream mining context, challenges associated to, e.g., volume can benefit from general-purpose big data solutions. These include, among others, scalable machine learning and online NLP algorithms, crowdsourcing approaches for data labeling, visualization advances, and visual analytics for the monitoring and interpretation of activities on social platforms.

Open Problems

Opinion stream mining is a rather young area. Open problems include:

- How to extend the traditional notion of "concept drift" so that it also cover changes in the feature space? How to design algorithms that detect such changes and adapt to them in an efficient way?
- How to distinguish between concept drift and "virtual drift" (Gama et al. 2014), i.e., between changes that do affect the decision boundary and changes that do not?

  Especially in an opinionated stream, many changes occur at each moment, e.g., new words appear, and the number of postings changes with the hour of the day, but not all of them require model adaption. How to design algorithms that recognize virtual drift and only adapt the model when true concept drift occurs?
- How to capture changes in the semantics and polarity of words?

  If a word's semantics or polarity change, how to inform existing resources (e.g., lexica like SentiWordNet) that a word's meaning and polarity are different for old documents than for recent ones?
- How to deal with label veracity in the stream? A promising approach is crowdsourcing, s is done, e.g., in CloudFlows (Kranjc et al. 2015). *Amazon Mechanical Turk* is a popular platform, where one can upload tasks for crowdsourcing. However, crowdsourcing has not been designed for learning and adaption on a fast stream, so solutions that also deal with stream velocity are necessary.

An associated open issue that can also be found in text stream mining, e.g., in the analysis of news streams, concerns the description of *bursts*. A burst is a rapid increase in social activity and may also be associated with a rapid change in the class priors and in the words being used to express polarity and to express facts. Do these changes disappear after the burst fades out, or do people take up the new words/expressions and use them also when they express opinions on other subjects? Does a burst lead to (more) permanent changes in the way people express opinions, on their perception toward a given entity, or on the topics they discuss?

Impact

Opinions have been always important for decision making. The opinion deluge we encounter nowadays mainly due to the WWW and the widespread usage of social networks is transforming business, society, and our own decisions on, e.g., what product to buy, which movie to watch, etc. Opinion (stream) mining offers solutions for automatically exploiting such sort of data for decision making, through, e.g., prediction models. Beyond its usage as a "standalone tool" for, e.g., polarity prediction, opinion (stream) mining has an impact on other areas of research, an example of which is the area of *recommenders:* next to the ratings typically used by recommenders, it is possible to also capitalize on the user reviews as more and more users also provide reviews on the rated items. These reviews are rich in information: they typically describe the aspects of the items that the users like/dislike. Further, if there are no ratings, they may be inferred from the reviews. A recent work in this area is McAuley and Leskovec (2013).

Cross-References

▶ Active Learning
▶ Concept Drift
▶ Co-training
▶ Incremental Learning

- ▸ Online Learning
- ▸ Semi-supervised Learning
- ▸ Sentiment Analysis and Opinion Mining

Recommended Reading

Some of the publications cited thus far elaborate on issues that were only briefly touched in this lemma. In Liu (2012), Bing Liu gives a thorough overview of sentiment analysis and opinion mining. For text classification methods, readers are referred to the recent book chapter of Aggarwal and Zhai (2014).

References

Aggarwal CC, Yu PS (2006) A framework for clustering massive text and categorical data. In: Proceedings of 6th SIAM international conference on data mining (SDM'06), Bethesda. SIAM, pp 479–483

Aggarwal C, Zhai C (2014) Text classification. In: Aggarwal C (ed) Data classification: algorithms and applications, chapter 11. Chapman & Hall/CRC, Boca Raton, pp 287–336

AlSumait L, Barbara D, Domeniconi C (2008) On-line LDA: adaptive topic models for mining text streams with applications to topic detection and tracking. In: Proceedings of 2008 IEEE conference on data mining (ICDM'08), Pisa. IEEE, pp 373–382

Bifet A, Frank E (2010) Sentiment knowledge discovery in Twitter streaming data. In: Proceedings of the 13th international conference on discovery science (DS'10), Canberra. Springer, pp 1–15

Bifet A, Gavaldà R (2009) Adaptive learning from evolving data streams. In: Proceedings of the 8th international symposium on intelligent data analysis: advances in intelligent data analysis VIII (IDA), Lyon. Springer, pp 249–260

Blei DM, Lafferty JD (2006) Dynamic topic models. In: Proceedings of 23rd international conference on machine learning (ICML'06), Pittsburgh, pp 113–120

Blum A, Mitchell T (1998) Combining labeled and unlabeled data with co-training. In: Proceedings of 11th conference on computational learning theory, Madison. ACM, pp 92–100

Chapelle O, Schölkopf B, Zien A (2006) Semi-supervised learning. MIT, Cambridge

Cheng Y, Chen Z, Liu L, Wang J, Agrawal A, Choudhary A (2013) Feedback-driven multiclass active learning for data streams. In: Proceedings of 22nd international conference on information and knowledge management (CIKM'13), San Fransisco, pp 1311–1320

Fu X, Yang K, Huang JZ, Cui L (2015) Dynamic non-parametric joint sentiment topic mixture model. Know-Based Syst 82(C):102–114

Gama J, Žliobaitė I, Bifet A, Pechenizkiy M, Bouchachia A (2014) A survey on concept drift adaptation. ACM Comput Surv 46(4):44:1–44:37

Gan H, Sang N, Huang R, Tong X, Dan Z (2013) Using clustering analysis to improve semi-supervised classification. Neurocomputing 101:290–298

Gohr A, Hinneburg A, Schult R, Spiliopoulou M (2009) Topic evolution in a stream of documents. In: SIAM data mining conference (SDM'09), Reno, pp 378–385

Gokulakrishnan B, Priyanthan P, Ragavan T, Prasath N, Perera A (2012) Opinion mining and sentiment analysis on a Twitter data stream. In: Proceedings of the 2012 international conference on advances in ICT for emerging regions (ICTer), Colombo, pp 182–188

Kranjc J, Smailovic J, Podpecan V, Grcar M, Znidarsic M, Lavrac N (2015) Active learning for sentiment analysis on data streams: methodology and workflow implementation in the ClowdFlows platform. Inf Process Manag 51(2):187–203

Liu B (2012) Sentiment analysis and opinion mining. Synth Lect Hum Lang Technol 5(1):1–167

Liu S, Li F, Li F, Cheng X, Shen H (2013) Adaptive co-training SVM for sentiment classification on tweets. In: Proceedings of 22nd international conference on information and knowledge management (CIKM'13), San Fransisco, pp 2079–2088

McAuley J, Leskovec J (2013) Hidden factors and hidden topics: understanding rating dimensions with review text. In: Proceedings of 7th ACM conference on recommender systems (RecSys'13), Hong Kong. ACM, pp 165–172

Saveski M, Grcar M (2011) Web services for stream mining: a stream-based active learning use case. In: Proceedings of the workshop "Planning to Learn and Service-Oriented Knowledge Discovery" at ECML PKDD 2011, Athens

Wagner S, Zimmermann M, Ntoutsi E, Spiliopoulou M (2015) Ageing-based multinomial naive bayes classifiers over opinionated data streams. In: European conference on machine learning and principles and practice of knowledge discovery in databases (ECMLPKDD'15), Porto, 07–11 Sept 2015. Volume 9284 of lecture notes in computer science. Springer International Publishing

Wang X, McCallum A (2006) Topics over time: a non-Markov continuous-time model of topical trends. In: Proceedings of 12th ACM SIGKDD international conference on knowledge discovery and data mining (KDD'06), Philadelphia, pp 424–433

Zhou S, Chen Q, Wang X (2013) Active deep learning method for semi-supervised sentiment classification. Neurocomputing 120:536–546

Zimmermann M, Ntoutsi E, Spiliopoulou M (2015a) Discovering and monitoring product features and the opinions on them with OPINSTREAM. Neurocomputing 150:318–330

Zimmermann M, Ntoutsi E, Spiliopoulou M (2015b) Incremental active opinion learning over a stream of opinionated documents. In: WISDOM’15 (workshop on issues of sentiment discovery and opinion mining) at KDD’15, Sydney

Zimmermann M, Ntoutsi E, Spiliopoulou M (2016) Extracting opinionated (sub)features from a stream of product reviews using accumulated novelty and internal re-organization. Inf Sci 329:876–899

Zliobaite I, Bifet A, Pfahringer B, Holmes G (2011) Active learning with evolving streaming data. In: Proceedings of ECML PKDD 2011, Athens. Volume 6913 of LNCS. Springer

Optimal Learning

▶ Bayesian Reinforcement Learning

Ordered Rule Set

▶ Decision List

Ordinal Attribute

An *ordinal attribute* classifies data into categories that can be ranked. However, the differences between the ranks cannot be calculated by arithmetic. See ▶ Attribute and ▶ Measurement Scales.

Out-of-Sample Data

Out-of-sample data are data that were not used to learn a model. ▶ Holdout evaluation uses out-of-sample data for evaluation purposes.

Out-of-Sample Evaluation

Definition

Out-of-sample evaluation refers to ▶ algorithm evaluation whereby the learned model is evaluated on ▶ out-of-sample data, which are data that were not used in the process of learning the model. Out-of-sample evaluation provides a less biased estimate of learning performance than ▶ in-sample evaluation. ▶ Cross validation, ▶ holdout evaluation and ▶ prospective evaluation are three main approaches to out-of-sample evaluation. Cross validation and holdout evaluation run risks of overestimating performance relative to what should be expected on future data, especially if the data set used is not a true random sample of the distribution on which the learned models are to be applied in the future.

Cross-References

▶ Algorithm Evaluation

Overall and Class-Sensitive Frequencies

The underlying idea for learning strategies processing ▶ missing attribute values relies on the class distribution; i.e., the class-sensitive frequencies are utilized. As soon as we substitute a missing value by a suitable one, we take the desired class of the example into consideration in order not to increase the noise in the data set. On the other hand, the overall (class-independent) frequencies are applied within classification.

Overfitting

Geoffrey I. Webb
Faculty of Information Technology, Monash University, Clayton, Melbourne, VIC, Australia

Synonyms

Overtraining

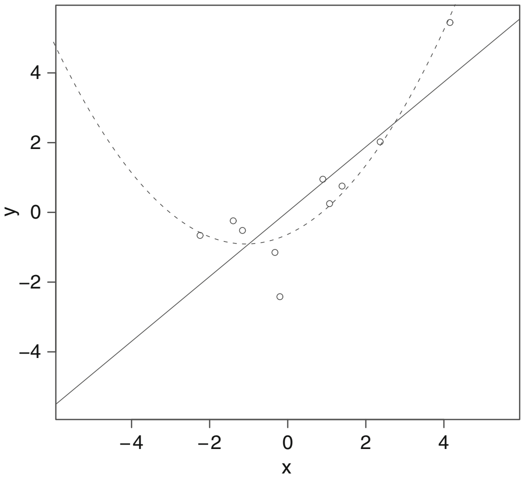
Definition

A model *overfits* the ▶ training data when it describes features that arise from noise or variance in the data, rather than the underlying distribution from which the data were drawn. Overfitting usually leads to loss of ▶ accuracy on ▶ out-of-sample data.

Discussion

In general there is a trade-off between the size of the space of distinct models that a learner can produce and the risk of overfitting. As the space of models between which the learner can select increases, the risk of overfitting will increase. However, the potential for finding a model that closely fits the true underling distribution will also increase. This can be viewed as one facet of the ▶ bias and variance trade-off.

Figure 1 illustrates overfitting. The points are drawn randomly from a distribution in which $y = x + \varepsilon$, where ε is random noise. The best single line fit to this distribution is $y = x$. ▶ Linear regression finds a model $y = 0.02044 + 0.92978 \times x$, shown as the solid line in Fig. 1. In contrast, second degree polynomial regression finds the model $-0.6311 + 0.5128 \times x + 0.2386 \times x^2$, shown as the dashed line. The space of second degree polynomial models is greater than that of linear models, and so the second degree polynomial more closely fits the example data, returning the lower ▶ squared error. However, the linear model more closely fits the true distribution and is more likely to obtain lower squared error on future samples.



Overfitting, Fig. 1 Linear and polynomial models fitted to random data drawn from a distribution for which the linear model is a better fit

While this example relates to ▶ regression, the same effect also applies to classification problems. For example, an overfitted ▶ decision tree may include splits that reflect noise rather than underlying regularities in the data.

The many approaches to avoiding overfitting include

- Using low variance learners;
- ▶ Minimum Description Length and ▶ Minimum Message Length techniques
- ▶ Pruning
- ▶ Regularization

Cross-References

▶ Bias Variance Decomposition
▶ Minimum Description Length Principle
▶ Minimum Message Length
▶ Pruning
▶ Regularization

Overtraining

▶ Overfitting

P

PAC Identification

▶ PAC Learning

PAC Learning

Thomas Zeugmann
Hokkaido University, Sapporo, Japan

Synonyms

Distribution-free learning; PAC identification; Probably approximately correct learning

Motivation and Background

A very important learning problem is the task of *learning a concept*. ▶ Concept learning has attracted much attention in learning theory. For having a running example, we look at humans who are able to distinguish between different "things," e.g., chair, table, car, airplane, etc. There is no doubt that humans have to learn how to distinguish "things." Thus, in this example, each concept is a thing. To model this learning task, we have to convert "real things" into *mathematical descriptions of things*. One possibility to do this is to fix some language to express a *finite* list of properties. Afterward, we decide which of these properties are relevant for the particular things we want to deal with and which of them have to be fulfilled or not to be fulfilled, respectively. The list of properties comprises qualities or traits such as "has four legs," "has wings," "is green," "has a backrest," "has a seat," etc. So these properties can be regarded as Boolean predicates, and, provided the list of properties is large enough, each thing can be described by a conjunction of these predicates. For example, a chair is described by "has four legs and has a backrest and has a seat and has no wings." Note that the color is not relevant and thus, "is green" has been omitted.

Assume that we have n properties, where n is a natural number. In the easiest case, we can denote the n properties by Boolean variables $x_1, \ldots, x_n$, where $range(x_j) \subseteq \{0, 1\}$ for $j = 1, \ldots, n$. The semantics is then obviously defined as follows. Setting $x_j = 1$ means property j is fulfilled, while $x_j = 0$ refers to property j is not fulfilled. Now, setting $\mathcal{L}_n = \{x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n\}$ (set of literals), we can express each thing as a conjunction of literals. As usual, we refer to any conjunction of literals as a *monomial*.

Therefore, formally we have as *learning domain* (also called ▶ instance space) the set of all Boolean vectors of length n, i.e., $\{0, 1\}^n$, and, in the learner's world, each thing (concept) is just a particular subset of $\{0, 1\}^n$. As far as our example is concerned, the concept chair is then the set of all Boolean vectors for which the monomial "has four legs and has a backrest and has a seat and has no wings" evaluates to 1.

Furthermore, it is usually assumed that the concept c to be learned (the target concept) is taken from a prespecified class $\mathcal{C}$ of possible

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

concepts called the *concept class*. In our example above, the concept class is the set of all concepts describable by a monomial. Consequently, we see that formally learning a concept is equivalent to identifying (exact or approximately) a set from a given set of possibilities by *learning* a suitable description (synonymously called representation) of it.

As in complexity theory, we usually assume that the representations are reasonable ones. Then they can be considered as strings over some fixed alphabet and the set of representations constitutes the ▸ representation language. Note that a concept may have more than one representation in a given representation language (and should have at least one) and that there may be different representation languages for one and the same concept class. For example, every Boolean function can be expressed as a ▸ conjunctive normal form (CNF) and as a ▸ disjunctive normal form (DNF), respectively. For a fixed representation language, the *size* of a concept is defined to be the length of a shortest representation for it. Since we are interested in a model of *efficient* learning, usually the following additional requirements are made: given any string over the underlying alphabet, one can decide in time polynomial in the length of the string whether or not it is a representation. Furthermore, given any element x from the underlying learning domain and a representation r for any concept, one can uniformly decide in time polynomial in the length of both inputs whether or not x belongs to the concept c described by r.

So, we always have a representation language used to define the concept class. As we shall see below, it may be advantageous to choose a possibly different representation language used by the learner. The class of all sets described by this representation language is called ▸ hypothesis space (denoted by $\mathcal{H}$), and the elements of it are said to be hypotheses (commonly denoted by h).

The *learner* is specified to be an algorithm. Further details are given below. We still have to specify the information source, the criterion of success, the hypothesis space, and the prior knowledge in order to define what PAC learning is.

The abbreviation PAC stands for *probably approximately correct* and the corresponding learning model has been introduced by Valiant (1984), while its name was dubbed by Angluin (1988). Valiant's (1984) pioneering paper triggered a huge amount of research the results of which are commonly called computational learning theory (see also the COLT and ALT conference series). Comprehensive treatises of this topic include Anthony and Biggs (1992), Kearns and Vazirani (1994), as well as Natarajan (1991).

Informally, this means that the learner has to find, on input, a randomly drawn set of labeled examples (called *sample*), with high probability a hypothesis such that the error of it is small. Here the error is measured with respect to the same probability distribution D with respect to which the examples are drawn.

Let $X \neq \emptyset$ be any learning domain and let $\mathcal{C} \subseteq \wp(X)$ be any nonempty concept class (here $\wp(X)$ denotes the power set of X). If X is infinite, we need some mild measure theoretic assumptions to ensure that the probabilities defined below exist. We refer to such concept classes as *well-behaved* concept classes. In particular, each $c \in \mathcal{C}$ has to be a Borel set. For a more detailed discussion, see Blumer et al. (1989).

Next, we formally define the *information source*. We assume any unknown probability distribution D over the learning domain X. No assumption is made concerning the nature of D and the learner has no knowledge concerning D. There is a *sampling oracle* $EX(\ )$, which has no input. Whenever $EX(\ )$ is called, it draws an element $x \in X$ according to D and returns the element x together with an indication of whether or not x belongs to the target concept c. Thus, every example returned by $EX(\ )$ may be written as $(x, c(x))$, where $c(x) = 1$ if $x \in c$ (positive examples) and $c(x) = 0$ otherwise (negative examples). If we make s calls to the example $EX(\ )$, then the elements $x_1, \dots, x_s$ are drawn independently from one another. Thus, the resulting probability distribution over all s-tuples of elements from X is the s-fold product distribution of D, i.e.,

$$\Pr(x_1, \ldots, x_s) = \prod_{i=1}^{s} D(x_i)\ , \qquad (1)$$

where $\Pr(A)$ denotes the probability of event A. Hence, the information source for a target concept c is any randomly drawn s-sample $S(c, \bar{x}) = (x_1, c(x_1), \ldots, x_s, c(x_s))$ returned by $EX(\ )$.

The *criterion of success*, i.e., *probably approximately correct* learning, is parameterized with respect to two quantities, the *accuracy parameter* ε, and the *confidence parameter* δ, where $\varepsilon, \delta \in (0, 1)$. Next, we define the *difference* between two sets $c,\ c' \subseteq X$ with respect to the probability distribution D as

$$d(c, c') = \sum_{x \in c \triangle c'} D(x),$$

where $c \triangle c'$ denotes the symmetric difference, i.e., $c \triangle c' = c \setminus c' \cup c' \setminus c$. We say that hypothesis h is an *ε-approximation* of a concept c, if $d(c, h) \leq \varepsilon$. A learner is *successful*, if it computes an ε-approximation of the target concept, and it should do so with probability at least $1 - \delta$.

The *hypothesis space* $\mathcal{H}$ is any set such that $\mathcal{C} \subseteq \mathcal{H}$, and the only *prior knowledge* is that the target concept is from the concept class.

A further important feature of the PAC learning model is the demand to learn efficiently. Usually, in the PAC learning model, the efficiency is measured with respect to the number of examples needed and the amount of computing time needed, and in both cases the requirement is to learn with an amount that is polynomial in the "size of the problem." In order to arrive at a meaningful definition, one has to discuss the problem size and, in addition, to look at the asymptotic difficulty of the learning problem. That is, instead of studying the complexity of some fixed learning problem, we always look at infinite sequences of similar learning problems. Such infinite sequences are obtained by allowing the size (dimension) of the learning domain to grow or by allowing the complexity of the concepts considered to grow. In both cases we use n to denote the relevant parameter.

Definition

A learning method $\mathcal{A}$ is said to *probably approximately correctly learn a target concept c with respect to a hypothesis space $\mathcal{H}$* and with sample complexity $s = s(\varepsilon, \delta)$ (or $s = s(\varepsilon, \delta, n)$), if for any distribution D over X and for all $\varepsilon, \delta \in (0, 1)$, it makes s calls to the oracle $EX(\ )$, and after having received the answers produced by $EX(\ )$ (with respect to the target c), it always stops and outputs a representation of a hypothesis $h \in \mathcal{H}$ such that

$$\Pr(d(c, h) \leq \varepsilon) \geq 1 - \delta.$$

A learning method $\mathcal{A}$ is said to *probably approximately correctly identify a target concept class $\mathcal{C}$ with respect to a hypothesis space $\mathcal{H}$* and with sample complexity $s = s(\varepsilon, \delta)$, if it probably approximately correctly identifies every concept $c \in \mathcal{C}$ with respect to $\mathcal{H}$ and with sample complexity s.

A learning method $\mathcal{A}$ is said to be *efficient*, if there exists a polynomial *pol* such that the running time of $\mathcal{A}$ and the number s of examples seen are at most $pol(1/\varepsilon, 1/\delta, n)$.

Remarks

This looks complicated, and so, some explanation is in order. First, the inequality

$$\Pr(d(c, h) \leq \varepsilon) \geq 1 - \delta$$

says that with high probability (quantified by δ), there is not too much difference (quantified by ε) between the conjectured concept (described by h) and the target c. Formally, let $\mathcal{A}$ be any fixed learning method, and let c be any fixed target concept. For any fixed $\varepsilon, \delta \in (0, 1)$, let $s = s(\varepsilon, \delta)$ be the actual sample size. We have to consider all possible outcomes of $\mathcal{A}$ when run on every labeled s-sample $S(c, \bar{x}) = (x_1, c(x_1), \ldots, x_s, c(x_s))$ returned by $EX(\ )$. Let $h(S(c, \bar{x}))$ be the hypothesis produced by $\mathcal{A}$ when processing $S(c, \bar{x})$. Then we have to consider the set W of all s-tuples over X such that $d(c, h(S(c, \bar{x}))) \leq \varepsilon$. The condition $\Pr(d(c, h) \leq \varepsilon) \geq 1 - \delta$ can now be formally rewritten as $\Pr(W) \geq 1 - \delta$. Clearly, one has to require that

$\Pr(W)$ is well defined. Note that the sample size is *not* allowed to depend on the distribution D.

To exemplify this approach, recall that our set of all concepts describable by a monomial over $\mathcal{L}_n$ refers to the set of all things. We consider a hypothetical learner (e.g., a student, a robot) that has to learn the concept of a chair. Imagine that the learner is told by a teacher whether or not particular things visible by the learner are instances of a chair. What things are visible depends on the environment the learner is in. The formal description of this dependence is provided by the unknown distribution D. For example, the learner might be led to a kitchen, a sitting room, a bookshop, a beach, etc. Clearly, it would be unfair to teach the concept of a chair in a bookshop and then testing the learning success at a beach. Thus, the learning success is measured with respect to the same distribution D with respect to which the sampling oracle has drawn its examples. However, the learner is required to learn with respect to any distribution. That is, independently of whether the learner is led to a kitchen, a bookshop, a sitting room, a beach, etc., it has to learn with respect to the place it has been led to. The sample complexity refers to the amount of information needed to ensure successful learning. Clearly, the smaller the required distance of the hypothesis produced and the higher the confidence desired, the more examples are usually needed. But there might be atypical situations. To have an extreme example, the kitchen the learner is led to turned out to be empty. Since the learner is required to learn with respect to a typical kitchen (described by the distribution D), it may well fail under this particular circumstance. Such failure has to be restricted to atypical situations, and this is expressed by demanding the learner to be successful with confidence $1 - \delta$.

This corresponds to real-life situations. For example, a student who has attended a course in learning theory might well suppose that she is examined in learning theory and not in graph theory. However, a good student, say in computer science, has to pass all examinations successfully, independently of the particular course attended. That is, she must successfully pass examinations in computability theory, complexity theory, cryptology, parallel algorithms, etc. Hence, she has to learn a whole concept class. The sample complexity refers to the time of interaction performed by the student and teacher. Also, the student may come up with a different representation of the concepts taught than the teacher. If we require $\mathcal{C} = \mathcal{H}$, then the resulting model is referred to as *proper* PAC learning.

The Finite Case

Having reached this point, it is natural to ask which concept classes are (efficiently) PAC learnable. We start with the finite case, i.e., learning domains X of finite cardinality. As before, the s-sample of c generated by $\bar{x}$ is denoted by $S(c, \bar{x}) = (x_1, c(x_1), \ldots, x_s, c(x_s))$. A hypothesis $h \in \mathcal{H}$ is called *consistent* for an s-sample $S(c, \bar{x})$, if $h(x_i) = c(x_i)$ for all $1 \leq i \leq s$. A learner is said to be *consistent* if all its outputs are consistent hypotheses. Then the following strategy (also known as ▸ Occam's razor) may be used to design a PAC learner:

(1) Draw a sufficiently large sample from the oracle $EX(\,)$, say s examples.
(2) Find some $h \in \mathcal{H}$ that is consistent with all the s examples drawn.
(3) Output h.

This strategy has a couple of remarkable features. First, provided the learner can find a consistent hypothesis, it allows for a uniform bound of the number of examples needed. That is,

$$s \geq \frac{1}{\varepsilon}\left(\ln|\mathcal{H}| + \ln\left(\frac{1}{\delta}\right)\right) \tag{2}$$

examples will always suffice (here $|S|$ denotes the cardinality of any set S).

The first insight obtained here is that increasing the confidence is exponentially cheaper than reducing the error.

Second, we see why we have to look at the asymptotic difficulty of the learning problem. If we fix $\{0, 1\}^n$ as learning domain and define $\mathcal{C}$ to be the set of all concepts describable by a Boolean function, then there are 2^{2^n} many concepts over $\{0,1\}^n$. Consequently, $\ln|\mathcal{H}| = O(2^n)$ resulting in a sample complexity that is for sure infeasible if $n \geq 50$. Thus, we set $X_n = \{0,1\}^n$, consider $\mathcal{C}_n \subseteq \wp(X_n)$, and study the relevant learning problem for $(X_n, \mathcal{C}_n)_{n\geq 1}$. So, finite means that all X_n are finite.

Third, using Inequality (2), it is not hard to see that the set of all concepts over $\{0,1\}^n$ that are describable by a monomial is efficiently PAC learnable. Let $\mathcal{H}_n$ be the set of all monomials containing each literal from $\mathcal{L}_n$ at most once plus the conjunction of all literals (denoted by m_{all}) (representing the empty concept). Since there are $3^n + 1$ monomials in $\mathcal{H}_n$, by (2), we see that $O(1/\varepsilon \cdot (n + \ln(1/\delta)))$ many examples suffice. Note that $2n$ is also an upper bound for the size of any concept from $\mathcal{H}_n$.

Thus it remains to deal with the problem to find a consistent hypothesis. The learning algorithm can be informally described as follows. After having received the s examples, the learner disregards all negative examples received and uses the positive ones to delete all literals from m_{all} that evaluate to 0 on at least one positive example. It then returns the conjunction of the literals not deleted from m_{all}. After a bit of reflection, one verifies that this hypothesis is consistent. This is essentially Haussler's (1987) Wholist algorithm and its running time is $O(1/\varepsilon \cdot (n^2 + \ln(1/\delta)))$. Also note that the particular choice of the representation for the empty concept was crucial here. It is worth noticing that the sample complexity is tight up to constant factors.

Using similar ideas one can easily show that the class of all concepts over $\{0,1\}^n$ describable by a k-CNF or k-DNF (where k is fixed) is efficiently PAC learnable by using as hypothesis space all k-CNF and k-DNF, respectively (cf. Valiant 1984). Note that a k-CNF is a conjunctive normal form in which each clause has at most k literals, and a k-DNF is a disjunctive normal form in which each monomial has at most k literals.

So, what can we say in general concerning the problem to find a consistent hypothesis? Answering this question gives us the insight to understand why it is sometimes necessary to choose a hypothesis space that is different from the target concept class. This phenomenon was discovered by Pitt and Valiant (1988). First, we look at the case where we have to efficiently PAC learn any $\mathcal{C}_n$ with respect to $\mathcal{C}_n$. Furthermore, an algorithm is said to *solve the consistency problem for* $\mathcal{C}_n$ if, on input any s-sample $S(c, \bar{x})$, where $c \subseteq X_n$, it outputs a hypothesis consistent with $S(c, \bar{x})$ provided there is one, and "there is no consistent hypothesis," otherwise.

Since we are interested in efficient PAC learning, we have to make the assumption that $|\mathcal{C}_n| \leq 2^{pol(n)}$ (cf. Inequality (2)). Also, it should be noted that for the proof of the following result, the requirement that $h(x)$ is polynomial time computable is essential (cf. our discussion of representations). Furthermore, we need the notion of an $\mathcal{RP}$-algorithm (randomized polynomial time). The input is any s-sample $S(c, \bar{x})$, where $c \subseteq X_n$ and the running time is uniformly bounded by a polynomial in the length of the input. In addition to its input, the algorithm can flip a coin in every step of its computation and then branch in dependence of the outcome of the coin flip. If there is no hypothesis consistent with $S(c, \bar{x})$, the algorithm must output "there is no consistent hypothesis," independently of the sequence of coin flips made. If there is a hypothesis consistent with $S(c, \bar{x})$, then the $\mathcal{RP}$-algorithm is allowed to fail with probability at most δ.

Interestingly, under the assumptions made above, then one can prove the following equivalence for efficient PAC learning.

PAC learning $\mathcal{C}_n$ with respect to $\mathcal{C}_n$ is equivalent to solving the consistency problem for $\mathcal{C}_n$ by an $\mathcal{RP}$-algorithm.

We continue by looking at the class of all concepts describable by a k-term DNF$_n$. A term is a conjunction of literals from $\mathcal{L}_n$, and a k-term DNF$_n$ is a disjunction of at most k terms. Consequently, there are $(3^n + 1)^k$ many k-term DNFs and thus the condition $|\mathcal{C}_n| \leq 2^{pol(n)}$ is fulfilled. Then one can show the following (see Pitt and Valiant 1988).

For all integers $k \geq 2$, if there is an algorithm that efficiently learns k-term DNF$_n$ *with respect to* k-term DNF$_n$, *then $\mathcal{RP} = \mathcal{NP}$.*

For a formal definition of the complexity classes $\mathcal{RP}$ and $\mathcal{NP}$, we refer the reader to Arora and Barak (2009). This result is proved by showing that deciding the consistency problem for k-term DNF$_n$ is $\mathcal{NP}$-complete for every $k \geq 2$. The difference between deciding and solving the consistency problem is that we only have to decide if there is a consistent hypothesis in k-term DNF$_n$. However, by the equivalence established above, we know that an efficient proper PAC learner for k-term DNF$_n$ can be transformed into an $\mathcal{RP}$-algorithm even solving the consistency problem. It should be noted that we currently do not know whether or not $\mathcal{RP} = \mathcal{NP}$ (only $\mathcal{RP} \subseteq \mathcal{NP}$ has been shown), but it is widely believed that $\mathcal{RP} \neq \mathcal{NP}$. On the other hand, it easy to see that every concept describable by a k-term DNF$_n$ is also describable by a k-CNF$_n$ (but not conversely). Thus, we can finally conclude that there is an algorithm that efficiently PAC learns k-term DNF$_n$ with respect to k-CNF$_n$.

For more results along this line of research, we refer the reader to Pitt and Valiant (1988), Blum and Singh (1990), and Jerrum (1994). As long as we do not have more powerful lower bound techniques allowing one to separate the relevant complexity classes $\mathcal{RP}$ and $\mathcal{NP}$ or $\mathcal{P}$ and $\mathcal{NP}$, no unconditional negative result concerning PAC learning can be shown. Another approach to show hardness results for PAC learning is based on cryptographic assumptions (cf., e.g., Kearns and Valiant 1989, 1994), and recently one has also tried to base cryptographic assumptions on the hardness of PAC learning (cf., e.g., Xiao (2009) and the references therein).

Further positive results comprise the efficient proper PAC learnability of rank k ▸ decision trees (cf. Ehrenfeucht and Haussler 1989) and of k-▸ decision lists for any fixed k (cf. Rivest 1987).

Finally, it must be noted that the bounds on the sample size obtained via Inequality (2) are *not* the best possible. Sometimes, better bounds can be obtained by using the ▸ VC dimension (see Inequality (4) below).

The Infinite Case

Let us start our exposition concerning infinite concept classes with an example due to Blumer et al. (1989). Consider the problem of learning concepts such as "medium built" animals. For the sake of presentation, we restrict ourselves to the parameters "weight" and "length." To describe "medium built," we use intervals "from-to." For example, a medium built cat might have a weight ranging from 3 to 7 kg and a length ranging from 25 cm to 50 cm. By looking at a finite database of randomly chosen animals giving their respective weight and length and their *classification* (medium built or not), we want to form a rule that *approximates* the true concept of "medium built" for each animal under consideration.

This learning problem can be formalized as follows. Let $X = \mathbb{E}^2$ be the two-dimensional Euclidean space, and let $\mathcal{C} \subseteq \wp(\mathbb{E}^2)$ be the set of all axis-parallel rectangles, i.e., products of intervals on the x-axis with intervals on the y-axis. Furthermore, let D be any probability distribution over X. Next we show that $\mathcal{C}$ is efficiently PAC learnable with respect to $\mathcal{C}$ by the following Algorithm **LR** (cf. Blumer et al. 1989):

Algorithm **LR**: "On input any $\varepsilon, \delta \in (0,1)$, call the oracle $EX(\ )$ s times, where $s = 4/\varepsilon \cdot \ln(4/\delta)$. Let $(r_1, c(r_1), r_2, c(r_2), \ldots, r_s, c(r_s))$ be the s-sample returned by $EX(\ )$, where $r_i = (x_i, y_i)$, $i = 1, \ldots s$.

$$\begin{aligned}\text{Compute } x_{\min} &= \min\{x_i \mid 1 \leq i \leq s,\ c(r_i) = 1\}\\ x_{\max} &= \max\{x_i \mid 1 \leq i \leq s,\ c(r_i) = 1\}\\ y_{\min} &= \min\{y_i \mid 1 \leq i \leq s,\ c(r_i) = 1\}\\ y_{\max} &= \max\{y_i \mid 1 \leq i \leq s,\ c(r_i) = 1\}\end{aligned}$$

Output $h = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$. In case there is no positive example, return $h = \emptyset$.
end."

It remains to show that Algorithm **LR** PAC learns the concept class $\mathcal{C}$ with respect to $\mathcal{C}$. Let $c = [a,b] \times [c,d]$ be the target concept. Since **LR** computes its hypothesis from positive examples, only, we get $h \subseteq c$. That is, h is consistent. We have to show that $d(c,h) \leq \varepsilon$ with probability at least $1 - \delta$. We distinguish the following cases.

Case 1. $D(c) \le \varepsilon$

Then $d(c,h) = \sum_{r \in c \triangle h} D(r) = \sum_{r \in c \setminus h} D(r) \le D(c) \le \varepsilon$.

Hence, in this case we are done.

Case 2. $D(c) > \varepsilon$

We define four minimal side rectangles within c that each cover an area of probability of at least $\varepsilon/4$. Let

Left $= [a,x] \times [c,d]$, where $x = \inf\{\tilde{x} \mid D([a,\tilde{x}] \times [c,d]) \ge \varepsilon/4\}$,

Right $= [z,b] \times [c,d]$, where $z = \inf\{\tilde{x} \mid D([\tilde{x},b] \times [c,d]) \ge \varepsilon/4\}$,

Top $= [a,b] \times [y,d]$, where $y = \inf\{\tilde{x} \mid D([a,b] \times [\tilde{x},d]) \ge \varepsilon/4\}$, and

Bottom $= [a,b] \times [c,t]$, where $t = \inf\{\tilde{x} \mid D([a,b] \times [c,\tilde{x}]) \ge \varepsilon/4\}$.

All those rectangles are contained in c, since $D(c) > \varepsilon$. If the sample size is s, the probability that a *particular* rectangle from $\{$*Left*, *Right*, *Top*, *Bottom*$\}$ contains no positive example is at most $(1-\varepsilon/4)^s$. Thus, the probability that *some* of those rectangles does not contain any positive example is at most $4(1-\varepsilon/4)^s$. Hence, incorporating $s = 4/\varepsilon \cdot \ln(4/\delta)$ gives

$$4(1-\varepsilon/4)^s < 4e^{-(\varepsilon/4)s} = 4e^{-\ln(4/\delta)} = \delta \,.$$

Therefore, with probability at least $1-\delta$, *each* of the four rectangles *Left*, *Right*, *Top*, and *Bottom* contains a positive example. Consequently, we get

$$d(c,h) = \sum_{r \in c \triangle h} D(r) \\ = \sum_{r \in c \setminus h} D(r) = D(c) - D(h) \,.$$

Furthermore, by construction

$$D(h) \ge D(c) - D(\mathit{Left}) - D(\mathit{Right}) \\ - D(\mathit{Top}) - D(\mathit{Bottom}) \ge D(c) - \varepsilon \,,$$

and hence $d(c,h) \le \varepsilon$.

Having reached this point, it is only natural to ask what makes infinite concept classes PAC learnable. Interestingly, there is a single parameter telling us whether or not a concept class is PAC learnable. This is the so-called Vapnik-Chervonenkis dimension commonly abbreviated as ▸ VC dimension. In our example of axis-parallel rectangles, the VC dimension of $\mathcal{C}$ is 4.

In order to state this result, we have to exclude trivial concept classes. A concept class $\mathcal{C}$ is said to be *trivial* if $|\mathcal{C}| = 1$ or $\mathcal{C} = \{c_1, c_2\}$ with $c_1 \cap c_2 = \emptyset$ and $X = c_1 \cup c_2$. A concept class $\mathcal{C}$ is called nontrivial if $\mathcal{C}$ is not trivial. Then Blumer et al. (1989) showed the following:

A nontrivial well-behaved concept class is PAC learnable if and only if its VC dimension is finite.

Moreover, if the VC dimension is finite, essentially the same strategy as in the finite case applies, i.e., it suffices to construct a consistent hypothesis from $\mathcal{C}$ (or from a suitably chosen hypothesis space $\mathcal{H}$ which must be well behaved) in random polynomial time.

So, it remains to estimate the sample complexity. Let d be the VC dimension of $\mathcal{H}$. Blumer et al. (1989) showed that

$$s \ge \max\left\{\frac{4}{\varepsilon}\log\frac{2}{\delta}, \ \frac{8d}{\varepsilon}\log\frac{13}{\varepsilon}\right\} \tag{3}$$

examples do suffice. This upper bound has been improved by Anthony et al. (1990) to

$$s \ge \frac{1}{\varepsilon(1-\sqrt{\varepsilon})}\left[\log\left(\frac{d/(d-1)}{\delta}\right) + 2d\log\left(\frac{6}{\varepsilon}\right)\right] . \tag{4}$$

Based on the work of Blumer et al. (1989) (and the lower bound they gave), Ehrenfeucht et al. (1988) showed that if $\mathcal{C}$ is nontrivial, then no learning function exists (for any $\mathcal{H}$) if $s < \frac{1-\varepsilon}{2\varepsilon}\log\frac{2}{\delta} + \frac{d-1}{64\varepsilon}$. These results give a precise characterization of the number of examples needed (apart from the gap of a factor of $O(\log \frac{1}{\varepsilon}))$ in terms of the VC dimension. Also note the sharp dichotomy here, either any consistent learner (computable or not) will do or no learner at all exists.

Two more remarks are in order here. First, these bounds apply to *uniform* PAC learning, i.e., the learner is taking ε and δ as input, only. As outlined in our discussion just before we gave the formal definition of PAC learning, it is meaningful to look at the asymptotic difficulty of learning. In the infinite case, we can increment the dimension n of the learning domain as we did in the finite case. We may set $X_n = \mathbb{E}^n$ and then consider similar concept classes $\mathcal{C}_n \subseteq \wp(X_n)$. For example, the concept classes similar to axis-parallel rectangles are axis-parallel parallelepipeds in $\mathbb{E}^n$. Then the VC dimension of $\mathcal{C}_n$ is $2n$, and all that is left is to add n as input to the learner and to express d as a function of n in the bound (4). Clearly, the algorithm **LR** can be straightforwardly generalized to a learner for $(X_n, \mathcal{C}_n)_{n \geq 1}$.

Alternatively, we use n to parameterize the complexity of the concepts to be learned. As an example consider $X = \mathbb{E}$ and let $\mathcal{C}_n$ be the set of all unions of at most n (closed or open) intervals. Then the ▸ VC dimension of $\mathcal{C}_n$ is $2n$, and one can design an efficient learner for $(X, \mathcal{C}_n)_{n \geq 1}$. Another example is obtained for $X = \mathbb{E}^2$ by defining $\mathcal{C}_n$ to be the class of all convex polygons having at most n edges (cf. Linial et al. 1991).

Second, all the results discussed so far are dealing with *static sampling*, i.e., any sample containing the necessary examples is drawn before any computation is performed. So, it is only natural to ask what can be achieved when *dynamic sampling* is allowed. In dynamic sampling mode, a learner alternates between drawing examples and performing computations. Under this sampling mode, even concept classes having an infinite VC dimension are learnable (cf. Linial et al. 1991 and the references therein). The main results in this regard are that enumerable concept classes and decomposable concept classes are PAC learnable when using dynamic sampling.

Let us finish the general exposition of PAC learning by pointing to another interesting insight, i.e., learning is in some sense data compression. As we have seen, finding consistent hypotheses is a problem of fundamental importance in the area of PAC learning. Clearly, the more expressive the representation language for the hypothesis space, the easier it may be to find a consistent hypothesis, but it may be increasingly difficult to say something concerning its accuracy (in machine learning this phenomenon is also known as the over-fitting problem). At this point, ▸ Occam's razor comes into play. If there is more than one explanation for a phenomenon, then Occam's razor requires to "prefer simple explanations." So, an Occam algorithm is an algorithm which, given a sample of the target concept, outputs a consistent and relatively simple hypothesis. That is, it is capable of some *data compression*. Let us first look at the Boolean case, i.e., $X_n = \{0,1\}^n$. Then an Occam algorithm is a randomized polynomial time algorithm $\mathcal{A}$ such that there is a polynomial p and a constant $\alpha \in [0,1)$ fulfilling the following demands:

For every $n \geq 1$, every target concept $c \in \mathcal{C}_n$ of size at most m and every $\varepsilon \in (0,1)$, on input any s-sample for c, algorithm $\mathcal{A}$ outputs with probability at least $1 - \varepsilon$ the representation of a consistent hypothesis from $\mathcal{C}_n$ having size at most $p(n, m, 1/\varepsilon) \cdot s^{\alpha}$.

So, the parameter $\alpha < 1$ expresses the amount of compression required. If we have such an Occam algorithm, then $(X_n, \mathcal{C}_n)$ is properly PAC learnable (cf. Blumer et al. 1987). The proof is based on the observations that a hypothesis with large error is unlikely to be consistent with a large sample and that there are only few short hypotheses. If we replace in the definition of an Occam algorithm the demand on the existence of a short hypothesis by the existence of a hypothesis space having a small VC dimension, then a similar result can be obtained for the continuous case (cf. Blumer et al. 1989). To a certain extent, the converse is also true, that is, under quite general conditions, PAC learnability implies the existence of an Occam algorithm. We refer the reader to Kearns and Vazirani (1994) for further details.

Variations

Further variations of PAC learning are possible and have been studied. So far, we have only considered one sampling oracle. Hence, a natural variation is to have two sampling oracles $EX_{+}(\ )$ and $EX_{-}(\ )$ and two distributions D_{+} and D_{-}, i.e., one for positive examples and one for negative examples. Clearly, further natural variations are possible. A larger number of them has been shown to be roughly equivalent and we refer the reader to Haussler et al. (1991) for details.

We continue with another natural variation that turned out to have a fundamental impact to the whole area of machine learning, i.e., weak learning.

Weak Learning

An interesting variation of PAC learning is obtained if we weaken the requirements concerning the confidence and the error. That is, instead of requiring the PAC learner to succeed for every ε and δ, one may relax this demand as follows. We only require the learner to succeed for $\varepsilon = 1/2 - 1/pol(n)$ (n is as above) and $\delta = 1/poly(n)$ (n is as above), where *pol* and *poly* are any two fixed polynomials. The resulting model is called *weak* PAC learning.

Quite surprisingly, Schapire (1990) could prove that every weak learner can be efficiently transformed into an ordinary PAC learner. While it is not too difficult to *boost* the confidence, *boosting* the error is much more complicated and has subsequently attracted a lot of attention. We refer the reader to Schapire (1990, 1999) as well as Kearns and Vazirani (1994) and the references therein for a detailed exposition. Interestingly enough, the techniques developed to prove the equivalence of weak PAC learnability and PAC learnability have an enormous impact to machine learning and may be subsumed under the title ▸ boosting.

Relations to Other Learning Models

Finally, we point out some relations of PAC learning to other learning models. Let us start with the mistake bound model also called online prediction model. The mistake bound model has its roots in ▸ inductive inference and was introduced by Littlestone (1988). It is conceptionally much simpler than the PAC model, since it does not involve probabilities. For the sake of presentation, we assume a finite learning domain X_n and any $\mathcal{C}_n \subseteq \wp(X_n)$ here.

In this model the following scenario is repeated indefinitely. The learner receives an instance x and has to predict $c(x)$. Then it is given the true label $c(x)$. If the learner's prediction was incorrect, then a *mistake* occurred. The learner is successful, if the total number of mistakes is finite. In order to make this learning problem nontrivial, one additionally requires that there is a polynomial *pol* such that for every $c \in \mathcal{C}_n$ and any ordering of the examples, the total number of mistakes is bounded by $pol(n, size(c))$. In the mistake bound model, a learner is said to be efficient if its running time per stage is uniformly polynomial in n and $size(c)$.

Then, the relation to PAC learning is as follows:

If algorithm $\mathcal{A}$ learns a concept class $\mathcal{C}$ in the mistake bound model, then $\mathcal{A}$ also PAC learns $\mathcal{C}$. Moreover, if $\mathcal{A}$ makes at most M mistakes, then the resulting PAC learner needs $\frac{M}{\varepsilon} \cdot \ln \frac{M}{\delta}$ many examples.

So, efficient mistake bound learning translates into efficient PAC learning.

Another interesting relation is obtained when looking at the ▸ query-based learning model, where the only queries allowed are equivalence queries. As pointed out by Angluin (1988, 1992), any learning method that uses equivalence queries only and achieves *exact* identification can be transformed into a PAC learner. The number of equivalence queries necessary to achieve success in the query learning model is polynomially related to the number of calls made to the sample oracle.

However, the converse is not true. This insight led to the definition of a *minimally adequate teacher* (cf. Angluin (1988) and the references therein). In this setting, the teacher answers equivalence queries and membership queries. Maas and Turán (1990) provide a detailed discussion of the relationship between the different models.

These results in turn led to another modification of the PAC model, where the learner is, in addition to the s-sample returned, also allowed to ask membership queries, i.e., PAC learning with membership queries. This and the original PAC learning model may be further modified by restricting the class of probability distributions, e.g., by considering PAC learning (with or without membership queries) with respect to the uniform distribution. Having the additional power of membership queries allowed for a series of positive polynomial time learnability results, e.g., the class of deterministic finite automata (cf. Angluin 1987), monotone DNF formulae (cf. Angluin 1988), polynomial size decision trees (cf. Bshouty 1993), and sparse multivariate polynomials over a field (cf. Schapire and Sellie 1996). Furthermore, Jackson (1997) showed the class of DNF formulae to be PAC learnable with membership queries under the uniform distribution, and Bshouty et al. (2004) presented a modification of Jackson's (1997) algorithm that substantially improves its asymptotic efficiency. Further variations of the PAC learning model are presented in Bshouty et al. (2005).

Let us finish this entry by mentioning that the PAC model has been criticized for two reasons. The first one is the independence assumption, that is, the requirement to learn with respect to any distribution. This is, however, also a very strong part of the theory, since it provides universal performance guarantees. Clearly, if one has additional information concerning the underlying distributions, one may be able to prove better bounds. The second reason is the "noise-free" assumption, i.e., the requirement to the sample oracle to return exclusively correct labels. Clearly, in practice we never have noise-free data. So, one has also studied learning in the presence of noise, and we refer the reader to Kearns and Vazirani (1994) as well as to conference series COLT and ALT for results along this line.

Cross-References

▶ Statistical Machine Translation
▶ Stochastic Finite Learning
▶ VC Dimension

Recommended Reading

Angluin D (1987) Learning regular sets from queries and counterexamples. Inf Comput 75(2):87–106

Angluin D (1988) Queries and concept learning. Mach Learn 2(4):319–342

Angluin D (1992) Computational learning theory: survey and selected bibliography. In: Proceedings of the 24th annual ACM symposium on theory of computing. ACM Press, New York, pp 351–369

Anthony M, Biggs N (1992) Computational learning theory. Cambridge tracts in theoretical computer science, vol 30. Cambridge University Press, Cambridge

Anthony M, Biggs N, Shawe-Taylor J (1990) The learnability of formal concepts. In: Fulk MA, Case J (eds) Proceedings of the third annual workshop on computational learning theory. Morgan Kaufmann, San Mateo, pp 246–257

Arora S, Barak B (2009) Computational complexity: a modern approach. Cambridge University Press, Cambridge

Blum A, Singh M (1990) Learning functions of k terms. In: Proceedings of the third annual workshop on computational learning theory. Morgan Kaufmann, San Mateo, pp 144–153

Blumer A, Ehrenfeucht A, Haussler D, Warmuth MK (1987) Occam's razor. Inf Process Lett 24(6):377–380

Blumer A, Ehrenfeucht A, Haussler D, Warmuth MK (1989) Learnability and the Vapnik-Chervonenkis dimension. J ACM 36(4):929–965

Bshouty NH (1993) Exact learning via the monotone theory. In: Proceedings of the 34rd annual symposium on foundations of computer science. IEEE Computer Society Press, Los Alamitos, pp 302–311

Bshouty NH, Jackson JC, Tamon C (2004) More efficient PAC-learning of DNF with membership queries under the uniform distribution. J Comput Syst Sci 68(1):205–234

Bshouty NH, Jackson JC, Tamon C (2005) Exploring learnability between exact and PAC. J Comput Syst Sci 70(4):471–484

Ehrenfeucht A, Haussler D (1989) Learning decision trees from random examples. Inf Comput 82(3):231–246

Ehrenfeucht A, Haussler D, Kearns M, Valiant L (1988) A general lower bound on the number of

examples needed for learning. In: Haussler D, Pitt L (eds) Proceedings of the 1988 workshop on computational learning theory (COLT'88), 3–5 Aug. MIT/Morgan Kaufmann, San Francisco, pp 139–154
Haussler D (1987) Bias, version spaces and Valiant's learning framework. In: Langley P (ed) Proceedings of the fourth international workshop on machine learning. Morgan Kaufmann, San Mateo, pp 324–336
Haussler D, Kearns M, Littlestone N, Warmuth MK (1991) Equivalence of models for polynomial learnability. Inf Comput 95(2):129–161
Jackson JC (1997) An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. J Comput Syst Sci 55(3):414–440
Jerrum M (1994) Simple translation-invariant concepts are hard to learn. Inf Comput 113(2):300–311
Kearns M, Valiant L (1994) Cryptographic limitations on learning Boolean formulae and finite automata. J ACM 41(1):67–95
Kearns M, Valiant LG (1989) Cryptographic limitations on learning Boolean formulae and finite automata. In: Proceedings of the 21st symposium on theory of computing. ACM Press, New York, pp 433–444
Kearns MJ, Vazirani UV (1994) An introduction to computational learning theory. MIT Press, Cambridge
Linial N, Mansour Y, Rivest RL (1991) Results on learnability and the Vapnik-Chervonenkis dimension. Inf Comput 90(1):33–49
Littlestone N (1988) Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. Mach Learn 2(4):285–318
Maas W, Turán G (1990) On the complexity of learning from counterexamples and membership queries. In: Proceedings of the 31st annual symposium on foundations of computer science (FOCS 1990), St. Louis, 22–24 Oct 1990. IEEE Computer Society, Los Alamitos, pp 203–210
Natarajan BK (1991) Machine learning: a theoretical approach. Morgan Kaufmann, San Mateo
Pitt L, Valiant LG (1988) Computational limitations on learning from examples. J ACM 35(4):965–984
Rivest RL (1987) Learning decision lists. Mach Learn 2(3):229–246
Schapire RE (1990) The strength of weak learnability. Mach Learn 5(2):197–227
Schapire RE (1999) Theoretical views of boosting and applications. In: Algorithmic learning theory, 10th international conference (ALT '99), Tokyo, Dec 1999, Proceedings. Lecture notes in artificial intelligence, vol 1720. Springer, pp 13–25
Schapire RE, Sellie LM (1996) Learning sparse multivariate polynomials over a field with queries and counterexamples. J Comput Syst Sci 52(2):201–213
Valiant LG (1984) A theory of the learnable. Commun ACM 27(11):1134–1142
Xiao D (2009) On basing $ZK \neq BPP$ on the hardness of PAC learning. In: Proceedings of the 24th annual IEEE conference on computational complexity (CCC 2009), Paris, 15–18 July 2009. IEEE Computer Society, Los Alamitos, pp 304–315

PAC-MDP Learning

▸ Efficient Exploration in Reinforcement Learning

Pairwise Classification

▸ Class Binarization

Parallel Corpus

A parallel corpus (pl. corpora) is a document collection composed of two or more disjoint subsets, each written in a different language, such that documents in each subset are translations of documents in each other subset. Moreover, it is required that the translation relation is known, i.e., that given a document in one of the subset (i.e., languages), it is known what documents in the other subset are its translations. The statistical analysis of parallel corpora is at the heart of most methods for ▸ cross-language text mining.

Part of Speech Tagging

▸ POS Tagging

Partially Observable Markov Decision Processes

Pascal Poupart
University of Waterloo, Waterloo, ON, Canada

Synonyms

Belief state Markov decision processes; Dual control; Dynamic decision networks; POMDPs

Definition

A partially observable Markov decision process (POMDP) refers to a class of sequential decision-making problems under uncertainty. This class includes problems with partially observable states and uncertain action effects. A POMDP is formally defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, b_0, h, \gamma \rangle$ where $\mathcal{S}$ is the set of states s, $\mathcal{A}$ is the set of actions a, $\mathcal{O}$ is the set of observations o, $T(s, a, s') = \Pr(s\prime|s, a)$ is the transition function indicating the probability of reaching s' when executing a in s, $Z(a, s', o') = \Pr(o'|a, s')$ is the observation function indicating the probability of observing o' in state s' after executing a, $R(s, a) \in \Re$ is the reward function indicating the (immediate) expected utility of executing a in s, $b_0 = \Pr(s_0)$ is the distribution over the initial state (also known as initial belief), h is the planning horizon (which may be finite or infinite), and $\gamma \in [0, 1]$ is a discount factor indicating by how much rewards should be discounted at each time step. Given a POMDP, the goal is to find a policy to select actions that maximize rewards over the planning horizon.

Motivation and Background

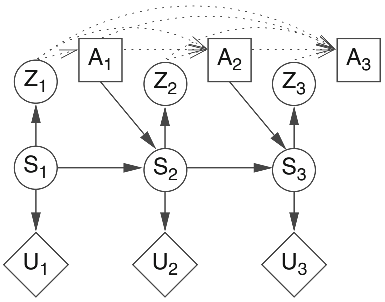
Partially observable Markov decision processes (POMDPs) were first introduced in the Operations Research community (Drake 1962; Aström 1965) as a framework to model stochastic dynamical systems and to make optimal decisions. This framework was later considered by the artificial intelligence community as a principled approach to planning under uncertainty (Kaelbling et al. 1998). Compared to other methods, POMDPs have the advantage of a well-founded theory. They can be viewed as an extension of the well-known, fully observable ▶ Markov decision process (MDP) model (Puterman 1994), which is rooted in probability theory, utility theory, and decision theory. POMDPs do not assume that states are fully observable, but instead that only part of the state features are observable, or more generally, that the observable features are simply correlated with the underlying states. This naturally captures the fact that in many real-world problems, the information available to the decision maker is often incomplete and typically measured by noisy sensors. As a result, the decision process is much more difficult to optimize. POMDP applications include robotics (Pineau and Gordon 2005), assistive technologies (Hoey et al. 2010), health informatics (Hauskrecht and Fraser 2010), spoken dialogue systems (Thomson and Young 2010), and fault recovery (Shani and Meek 2009).

Structure of Model and Solution Algorithms

We describe below the POMDP model, some policy representations, the properties of optimal value functions, and some solution algorithms.

POMDP Model

Figure 1 shows the graphical representation of a POMDP, using the notation of influence diagrams: circles denote random variables (e.g., state variables S_t and observation variables O_t), squares denote decision variables (e.g., action variables A_t), and diamonds denote utility variables (e.g., U_t's). The variables are indexed by time and grouped in time slices, reflecting the fact that each variable may take a different value at each time step. Arcs indicate how nodes influence each other over time. There are two types of arcs: probabilistic and informational arcs. Arcs pointing to a chance node or a utility node indicate

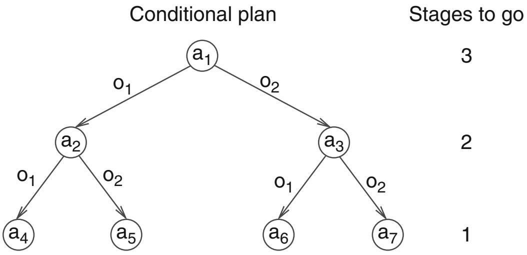


Partially Observable Markov Decision Processes, Fig. 1 POMDP represented as an influence diagram a probabilistic dependency between a child and its parents, whereas arcs pointing to a decision node indicate the information available to the decision maker (i.e., which nodes are observable at the time of each decision). Probabilistic dependencies for the state and observation variables are quantified by the conditional distributions $\Pr(S_{t+1}|S_t, A_t)$ and $\Pr(O_{t+1}|S_{t+1}, A_t)$, which correspond to the transition and observation functions. Note that the initial state variable S_0 does not have any parent, hence its distribution $\Pr(S_0)$ is unconditioned and corresponds to the initial belief b_0 of the decision maker. Probabilistic dependencies for the utility variables are also quantified by a conditional distribution $\Pr(U_t|S_t, A_t)$ such that its expectation $\sum_u \Pr(u|S_t,\ A_t)u\ =\ R(S_t,\ A_t)$ corresponds to the reward function.

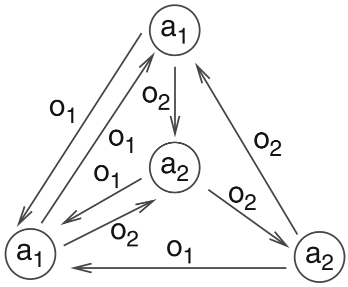
Fully observable MDPs are a special case of POMDPs since they arise when the observation function deterministically maps each state to a different unique observation. POMDPs can also be viewed as ▸ hidden Markov models (HMMs) (Rabiner 1989) extended with decision and utility nodes since the transition and observation distributions essentially define an HMM. POMDPs also correspond to a special case of decision networks called *dynamic decision networks* (Buede 1999) where it is assumed that the transition, observation, and reward functions are *stationary* (i.e., they do not depend on time) and *Markovian* (i.e., the parents of each variable are in the same time slice or immediately preceding time slice).

Policies

Given a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, b_0, h, \gamma \rangle$ specifying a POMDP, the goal is to find a policy π to select actions that maximize the rewards. The informational arcs indicate that each action *at* can be selected based on the history of past actions and observations. Hence, in its most general form, a policy $\pi : \langle b_0, h_t \rangle \rightarrow a_t$ is a mapping from initial beliefs b_0 and histories $h_t = \langle o_0, a_0, o_1, a_1, \ldots, o_{t-1}, a_{t-1}, o_t$r to actions a_t. For a fixed initial belief, the mapping can be represented by a tree such as the one in Fig. 2. We will refer to such policy trees as conditional plans since in general a policy may consist of several conditional plans for different initial beliefs. The execution of a conditional plan follows a branch from the root to some leaf by executing the actions of the nodes traversed and following the edges labeled by the observations received.



Partially Observable Markov Decision Processes, Fig. 2 Three representation of a three-step conditional plan



Partially Observable Markov Decision Processes, Fig. 3 Finite state controller for a simple POMDP with two actions and two observations

Unfortunately, as the number of steps increases, the number of histories grows exponentially and it is infeasible to represent mappings over all such histories. Furthermore, infinite-horizon problems require mappings over arbitrarily long histories, which limit the use of trees to problems with a short horizon. Note, however, that it is possible to have mappings over infinite *cyclic* histories. Such mappings can be represented by a *finite state controller* (Hansen 1997), which is essentially a cyclic graph of nodes labeled by actions and edges labeled by observations (see Fig. 3 for an example). Similar to conditional plans, finite state controllers are executed by starting at an initial node, executing the actions of the nodes traversed, and following the edges of the observations received.

Alternatively, it is possible to summarize histories by a sufficient statistic that encodes all the relevant information from previous actions and observations for planning purposes. Recall that the transition, reward, and observation functions exhibit the Markov property, which means that the outcome of future states, rewards, and observations depend only on the current state and action. If the decision maker knew the current state of the world, then she would have all the desired information to make an optimal action choice. Thus, histories of past actions and observations are only relevant to the extent that they provide information about the current state of the world. Let b_t be the belief of the decision maker about the state of the world at time step t, which we represent by a probability distribution over the state space $\mathcal{S}$. Using Bayes theorem (see Bayes Rules), one can compute the current belief b_t from the previous belief b_{t-1}, previous action a_{t-1}, and current observation o_t:

$$b_t(s') = k \sum_{s \in \mathcal{S}} b_{t-1}(s) \Pr(s'|s, a_{t-1}) \Pr(o_t|a_{t-1}, s') \tag{1}$$

where k denotes a normalizing constant. Hence, a policy π can also be represented as a mapping from beliefs b_t to actions a_t. While this gets around the exponentially large number of histories, the space of beliefs is an $|\mathcal{S}| - 1$-dimensional continuous space, which is also problematic. However, a key result by Smallwood and Sondik (1973) allows us to circumvent the continuous nature of the belief space. But first, let us introduce value functions and then discuss Smallwood and Sondik's solution.

Value Functions

Given a set of policies, we need a mechanism to evaluate and compare them. Roughly speaking, the goal is to maximize the amount of reward earned over time. This loosely defined criterion can be formalized in several ways: one may wish to maximize *total* (accumulated) or *average* reward, *expected* or *worst-case* reward, *discounted* or *undiscounted* reward. The rest of this article assumes an *expected total discounted* reward criterion, since it is by far the most popular in the literature. We define the value $V^\pi(b_0)$ of executing some policy π starting at belief b_0 to be the expected sum of the discounted rewards earned at each time step:

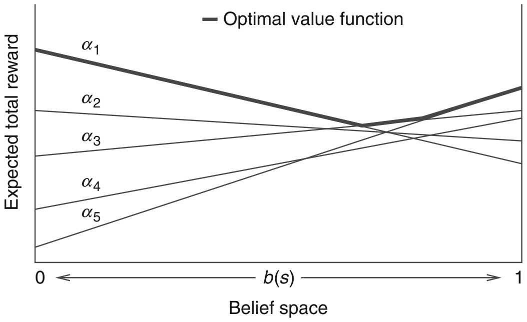
$$V^\pi(b_0) = \sum_{t=0}^{h} \gamma^t \sum_{s \in \mathcal{S}} b_t(s) R(s, \pi, (b_t)) \tag{2}$$

where $\pi(b_t)$ denotes the action prescribed by policy π at belief b_t. A policy π^* is optimal when its value function V^* is at least as high as any other policy for all beliefs (i.e., $V^*(b) \geq V^\pi(b) \forall b$).

As with policies, representing a value function can be problematic because its domain is an $(|\mathcal{S}| - 1)$-dimensional continuous space corresponding to the belief space. However, Smallwood and Sondik (1973) showed that optimal value functions for finite-horizon POMDPs are piecewise-linear and convex. The value of executing a conditional plan from any state is constant. If we do not know the precise underlying state, but instead we have a belief corresponding to a distribution over states, then the value of the belief is simply a weighted average (according to b) of the values of the possible states. Thus, the value function $V^\beta(b)$ of a conditional plan β is linear with respect to b. This means that $V^\beta(b)$ can be represented by a vector α_β of size $|\mathcal{S}|$ such that $V^\beta(b) = \Sigma_s b(s) \alpha_\beta(s)$.

For a finite horizon h, an optimal policy π^h consists of the best conditional plans for each initial belief. More precisely, the best conditional plan β^* for some belief b is the one that yields the highest value: $\beta^* = \operatorname{argmax}_\beta V^\beta(b)$. Although there are uncountably many beliefs, the set of h-step conditional plans is finite and therefore an h-step optimal value function can be represented by a finite collection Γ^h of α-vectors. For infinite horizon problems, the optimal value function may require an infinite number of α-vectors.

Figure 4 shows an optimal value function for a simple two-state POMDP. The horizontal axis represents the belief space and the vertical axis indicates the expected total reward. Assuming the two world states are s and $\bar{s}$, then a belief is completely determined by the probability of s. Therefore, the horizontal axis represents a continuum of beliefs determined by the probability



Partially Observable Markov Decision Processes, Fig. 4 Geometric view of value function

$b(s)$. Each line in the graph is an α-vector, which corresponds to the value function of a conditional plan. The upper surface of those α-vectors is a piecewise-linear and convex function corresponding to the optimal value function $V^* = \max_{\alpha \in \Gamma^h} \alpha(b)$.

Note that an optimal policy can be recovered from the optimal value function represented by a set Γ of α-vector. Assuming that an action is stored with each α-vector (this would typically be the root action of the conditional plan associated with each α-vector), then the decision maker simply needs to look up the maximal α-vector for the current belief to retrieve the action. Hence, value functions represented by a set of α-vectors, each associated with an action, implicitly define a mapping from beliefs to actions.

Optimal value functions also satisfy *Bellman's equation*

$$V^{h+1}(b) = \max_a R(b,a) + \gamma \sum_{o'} \Pr(o'|b,a) V^h(b^{ao'}) \quad (3)$$

where $R(b,a) = \sum_s b(s)R(s,a)$, $\Pr(s'|s,a)$ $\Pr(o'|s',a)$, and $b^{ao'}$ is the updated belief after executing a and observing b according to Bayes theorem (Eq. 1). Intuitively, this equation says that the optimal value for $h+1$ steps to go consists of the highest sum of the current reward with the future rewards for the remaining h steps. Since we do not know exactly what rewards will be earned in the future, an expectation (with respect to the observations) is used to estimate future rewards. For discounted infinite horizon problems, the optimal value function V^* is a fixed point of Bellman's equation:

$$V^*(b) = \max_a R(b,a) + \gamma \sum_{o'} \Pr(o'|b,a) V^*(b^{ao'})$$

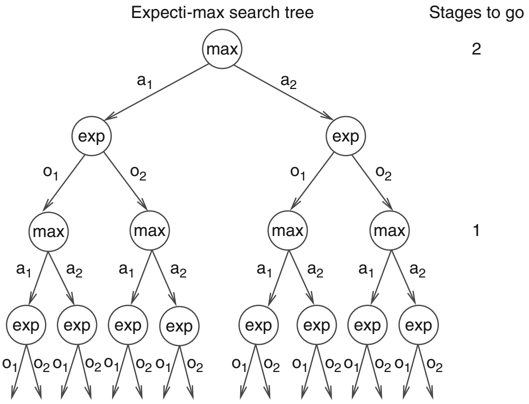
Solution Algorithms

There are two general classes of solution algorithms to optimize a policy. The first class consists of *online* algorithms that plan while executing the policy by growing a search tree. The second class consists of *offline* algorithms that precompute a policy which can be executed with minimal online computation. In practice, it is best to combine online and offline techniques since we may as well obtain the best policy possible in an offline phase and then refine it with an online search at execution time.

Forward Search

Online search techniques generally optimize a conditional plan for the current belief by performing a forward search from that belief. They essentially build an *expecti-max* search tree such that expectations over observations and maximizations over actions are performed in alternation. Figure 5 illustrates such a tree for a two-step horizon (i.e., two alternations of actions and observations). An optimal policy is obtained by computing the beliefs associated with each node in a forward pass, followed by a backward pass that computes the optimal value at each node. A recursive form of this approach is described in Algorithm 1. Beliefs are propagated forward according to Bayes theorem, while rewards are accumulated backward according to Bellman's equation.

Since the expecti-max search tree grows exponentially with the planning horizon h, in practice, the computation can often be simplified by pruning suboptimal actions by branch and bound and sampling a small set of observations instead of doing an exact expectation (Ross et al. 2008). Also, the depth of the search can be reduced by using an approximate value function at the leaves instead of 0.



Partially Observable Markov Decision Processes, Fig. 5 Two-step expecti-max search tree

Algorithm 1 Forward search

```
Inputs: Belief $b$ and horizon $h$
Outputs: Optimal value $V^*$.
if $h = 0$ then
  $V^* \leftarrow 0$
else
  for all $a, o$ do
    $b^{ao'}(s') \leftarrow k \sum_s b(s)\Pr(s'|s,a)\Pr(o'|s',a')\forall s'$
    $V^{ao'} \leftarrow forward\ Search(b^{ao'}, h-1)$
  end for
  $V^* \leftarrow \max_a R(b,a) + \gamma \sum_{o'} \Pr(o'|b,a) V^{ao'}$
end if
```

The value functions computed by offline techniques can often be used for this purpose.

Value Iteration

Value iteration algorithms form an important class of offline algorithms that iteratively estimate the optimal value function according to Bellman's equation (3). Most algorithms exploit the piecewise-linear and convex properties of optimal value functions to obtain a finite representation. In other words, optimal value functions Vh are represented by a set Γ^h of α-vectors that correspond to conditional plans. Algorithm 2 shows how to iteratively compute Γ^t by dynamic programming for an increasing number of time steps t.

Algorithm 2 Value iteration

```
Inputs: horizon $h$
Outputs: Optimal value function $\Gamma^h$.
$\Gamma^0 \leftarrow \{0\}$
for $t = 1$ to $h$ do
  for all $a \in \mathcal{A}, < \alpha_1, \ldots, \alpha_{|\mathcal{O}|} > \in (\Gamma^{t-1})^{|\mathcal{O}|}$ do
    $\alpha'(s) \leftarrow R(s,a) +$
    $\gamma \sum_{o',s'} \Pr(s'|s,a)\Pr(o'|s',a)\alpha_{o'}(s')\forall s$
    $\Gamma^t \leftarrow \Gamma^t \cup \{\alpha'\}$
    end for
  end for
```

Algorithm 3 Point based value iteration

```
Inputs: Horizon $h$ and set of beliefs $\mathcal{B}$
Outputs: Value function $\Gamma^h$.
$\Gamma^0 \leftarrow \{0\}$
for $t = 1$ to $h$ do
  for all $b \in \mathcal{B}$ do
    for all $a \in \mathcal{A}, o' \in \mathcal{O}$ do
      $b^{ao'}(s') \leftarrow k \sum_s b(s)\Pr(s'|s,a)\Pr(o'|s',a)\forall s'$
      $\alpha^{ao'} \leftarrow \text{argmax}_{alpha \in \Gamma^{t-1}} \alpha(b^{ao'})$
    end for
    $a^* \leftarrow \text{argmax}_a R(b,a) + \gamma \sum_{o'} \Pr(s'|s,a)\Pr\alpha^{ao'}$
    $\alpha'(s) R(s,a) + \gamma \sum_{o',s'} \Pr(s'|s,a)\Pr(o'|s',a)$
    $\alpha_{o'}(s')\forall s$
    $\Gamma^t \leftarrow \Gamma^t \cup \{\alpha'\}$
  end for
end for
```

Unfortunately, the number of α-vectors in each Γ^t increases exponentially with $\mathcal{O}$ and doubly exponentially with t. While several approaches can be used to prune α-vectors that are not maximal for any belief, the number of α-vectors still grows exponentially for most problems. Instead, many approaches compute a parsimonious set of α-vectors, which defines a lower bound on the optimal value function. The class of *point-based value iteration* (Pineau et al. 2006) algorithms computes the maximal α-vectors only for a set $\mathcal{B}$ of beliefs. Algorithm 3 describes how the parsimonious set Γ^h of α-vectors associated with a given set $\mathcal{B}$ of beliefs can be computed in time linear with h and $|\mathcal{O}|$ by dynamic programming. Most point-based techniques differ in how they choose $\mathcal{B}$ (which may vary at each iteration), but the general rule of thumb is to include beliefs reachable from the initial belief b_0 since these are the beliefs that are likely to be encountered at execution time.

Policy Search

Another important class of offline algorithms consists of policy search techniques. These techniques search for the best policy in a predefined space of policies. For instance, finite state controllers are a popular policy space due to their generality and simplicity. The search for the best (stochastic) controller of N nodes can be formulated as a non-convex quadratically constrained optimization problem (Amato et al. 2007):

$$\max_{x,y,z} \sum_s b_0(s)\, \underbrace{\alpha_0(s)}_{x}$$

$$\text{s.t. } \underbrace{\alpha_n(s)}_{x} = \sum_a [\underbrace{\Pr(a|n)}_{y}\, R(S,a)$$

$$+\gamma \sum_{s',0',n'} \Pr(s'|s,a)$$

$$\Pr(0'|s',a)\, \underbrace{\Pr(a,n'|n,0')}_{z}\, \underbrace{\alpha_{n'}(s')}_{x}]\ \forall s,n$$

$$\underbrace{\Pr(a,n'|n,0')}_{x} \geq 0 \forall a,n',n,0'$$

$$\sum_{n'a} \underbrace{\Pr(a,n'|n,0)}_{z} = 1 \forall n,0$$

$$\sum_{n'} \underbrace{\Pr(a,n'|n,0')}_{z} = \underbrace{\Pr(a|n)}_{y}\ \forall a,n,0'$$

The variables of the optimization problem are the α-vectors and the parameters of the controller ($\Pr(a|n)$ and $\Pr(a,\ n'|n,o')$). Here, $\Pr(a|n)$ is the action distribution for each node n and $\Pr(a,\ n'|n,o') = \Pr(a|n)\Pr(n'|a,n,o')$ is the product of the action distribution and successor node distribution for each n,o'-pair. While there does not exist any algorithm that reliably finds the global optimum due to the non-convex nature of the problem, several techniques can be used to find locally optimal policies, including sequential quadratic programming, bounded policy iteration, expectation maximization, stochastic local search, and gradient descent.

Related Work

Although this entry assumes that states, actions, and observations are defined by a single variable, multiple variables can be used to obtain a *factored POMDP* (Boutilier and Poole 1996). As a result, the state, observation, and action spaces often become exponentially large. Aggregation (Shani et al. 2008; Sim et al. 2008) and compression techniques (Poupart and Boutilier 2004; Roy et al. 2005) are then used to speed up computation. POMDPs can also be defined for problems with continuous variables. The piecewise-linear and convex properties of optimal value functions still hold in continuous spaces, which allows value iteration algorithms to be easily extended to continuous POMDPs (Porta et al. 2006). When a planning problem can naturally be thought as a hierarchy of subtasks, *hierarchical POMDPs* (Theocharous and Mahadevan 2002; Pineau et al. 2003; Toussaint et al. 2008) can be used to exploit this structure.

In this article, we also assumed that the transition, observation, and reward functions are known, but in many domains they may be (partially) unknown and therefore the decision maker needs to learn about them while acting. This is a problem of *reinforcement learning*. While several policy search techniques have been adapted to simultaneously learn and act (Meuleau et al. 1999; Aberdeen and Baxter 2002), it turns out that one can treat the unknown parameters of the transition, observation, and reward functions as hidden state variables, which lead to a *Bayes-adaptive POMDP* (Ross et al. 2007; Poupart and Vlassis 2008). We also assumed a single decision maker, however POMDPs have been extended for multiagent systems. In particular, *decentralized POMDPs* (Amato et al. 2009) can model multiple cooperative agents that share a common goal and *interactive POMDPs* (Gmytrasiewicz and Doshi 2005) can model multiple competing agents.

Cross-References

▶ Markov Decision Processes

Recommended Reading

Aberdeen D, Baxter J (2002) Scalable internal-state policygradient methods for POMDPs. In: International conference on machine learning, Sydney, pp 3–10

Amato C, Bernstein DS, Zilberstein S (2009) Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. J Auton Agents Multiagent Syst 21:293–320

Amato C, Bernstein DS, Zilberstein S (2007) Solving POMDPs using quadratically constrained linear programs. In: International joint conferences on artificial intelligence, Hyderabad, pp 2418–2424

Aström KJ (1965) Optimal control of Markov decision processes with incomplete state estimation. J Math Anal Appl 10:174–2005

Boutilier C, Poole D (1996) Computing optimal policies for partially observable decision processes using compact representations. In: Proceedings of the thirteenth national conference on artificial intelligence, Portland, pp 1168–1175

Buede DM (1999) Dynamic decision networks: an approach for solving the dual control problem. Spring INFORMS, Cincinnati

Drake A (1962) Observation of a Markov Process through a noisy channel. PhD thesis, Massachusetts Institute of Technology

Hansen E (1997) An improved policy iteration algorithm for partially observable MDPs. In: Neural information processing systems, Denver, pp 1015–1021

Hauskrecht M, Fraser HSF (2010) Planning treatment of ischemic heart disease with partially observable Markov decision processes. Artif Intell Med 18:221–244

Hoey J, Poupart P, von Bertoldi A, Craig T, Boutilier C, Mihailidis A (2010) Automated handwashing assistance for persons with dementia using video and a partially observable Markov decision process. Comput Vis Image Underst 114:503–519

Kaelbling LP, Littman M, Cassandra A (1998) Planning and acting in partially observable stochastic domains. Artif Intell 101:99–134

Meuleau N, Peshkin L, Kim K-E, Kaelbling LP (1999) Learning finite-state controllers for partially observable environments. In: Uncertainty in artificial intelligence, Stockholm, pp 427–436

Pineau J, Gordon G (2005) POMDP planning for robust robot control. In: International symposium on robotics research, San Francisco, pp 69–82

Pineau J, Gordon GJ, Thrun S (2003) Policy-contingent abstraction for robust robot control. In: Uncertainty in artificial intelligence, Acapulco, pp 477–484

Pineau J, Gordon G, Thrun S (2006) Anytime point-based approximations for large POMDPs. J Artif Intell Res 27:335–380

Gmytrasiewicz PJ, Doshi P (2005) A framework for sequential planning in multi-agent settings. J Artif Intell Res 24:49–79

Porta JM, Vlassis NA, Spaan MTJ, Poupart P (2006) Point-based value iteration for continuous POMDPs. J Mach Learn Res 7:2329–2367

Poupart P, Boutilier C (2004) VDCBPI: an approximate scalable algorithm for large POMDPs. In: Neural information processing systems, Vancouver, pp 1081–1088

Poupart P, Vlassis N (2008) Model-based Bayesian reinforcement learning in partially observable domains. In: International symposium on artificial intelligence and mathematics (ISAIM), Fort Lauderdale

Puterman ML (1994) Markov decision processes. Wiley, New York

Rabiner LR (1989) A tutorial on hidden Markov models and selected applications in speech recognition. Proc IEEE 77:257–286

Ross S, Chaib-Draa B, Pineau J (2007) Bayes-adaptive POMDPs. In: Advances in neural information processing systems (NIPS), Vancouver

Ross S, Pineau J, Paquet S, Chaib-draa B (2008) Online planning algorithms for POMDPs. J Artif Intell Res 32:663–704

Roy N, Gordon GJ, Thrun S (2005) Finding approximate POMDP solutions through belief compression. J Artif Intell Res 23:1–40

Shani G, Meek C (2009) Improving existing fault recovery policies. In: Neural information processing systems, Vancouver

Shani G, Brafman RI, Shimony SE, Poupart P (2008) Efficient ADD operations for point-based algorithms. In: International conference on automated planning and scheduling, Sydney, pp 330–337

Sim HS, Kim K-E, Kim JH, Chang D-S, Koo M-W (2008) Symbolic heuristic search value iteration for factored POMDPs. In: Twenty-third national conference on artificial intelligence (AAAI), Chicago, pp 1088–1093

Smallwood RD, Sondik EJ (1973) The optimal control of partially observable Markov decision processes over a finite horizon. Oper Res 21:1071–1088

Theocharous G, Mahadevan S (2002) Approximate planning with hierarchical partially observable Markov decision process models for robot navigation. In: IEEE international conference on robotics and automation, Washington, DC, pp 1347–1352

Thomson B, Young S (2010) Bayesian update of dialogue state: a POMDP framework for spoken dialogue systems. Comput Speech Lang 24:562–588

Toussaint M, Charlin L, Poupart P (2008) Hierarchical POMDP controller optimization by likelihood maximization. In: Uncertainty in artificial intelligence, Helsinki, pp 562–570

Particle Swarm Optimization

James Kennedy
U.S. Bureau of Labor Statistics, Washington,
DC, USA

The Canonical Particle Swarm

The particle swarm is a population-based stochastic algorithm for optimization which is based on social–psychological principles. Unlike ▸evolutionary algorithms, the particle swarm does not use selection; typically, all population members survive from the beginning of a trial until the end. Their interactions result in iterative improvement of the quality of problem solutions over time.

A numerical vector of D dimensions, usually randomly initialized in a search space, is conceptualized as a point in a high-dimensional Cartesian coordinate system. Because it moves around the space testing new parameter values, the point is well described as a particle. Because a number of them (usually $10 < N < 100$) perform this behavior simultaneously, and because they tend to cluster together in optimal regions of the search space, they are referred to as a *particle swarm*.

Besides moving in a (usually) Euclidean problem space, particles are typically enmeshed in a topological network that defines their communication pattern. Each particle is assigned a number of neighbors to which it is linked bidirectionally.

The most common type of implementation defines the particles' behaviors in two formulas. The first adjusts the velocity or step size of the particle, and the second moves the particle by adding the velocity to its previous position.

On each dimension d:

$$v_{id}^{(t+1)} \leftarrow \alpha v_{id}^{(t)} + U(0,\beta)\left(p_{id} - x_{id}^{(t)}\right) + U(0,\beta)\left(p_{gd} - x_{id}^{(t)}\right) \quad (1)$$

$$x_{id}^{(t+1)} \leftarrow x_{id}^{(t)} + v_{id}^{(t+1)} \quad (2)$$

where i is the target particle's index, d is the dimension, $\vec{x}_i$ is the particle's position, $\vec{v}_i$ is the velocity, $\vec{p}_i$ is the best position found so far by i, g is the index of i's best neighbor, α and β are constants, and $U(0, \beta)$ is a uniform random number generator.

Though there is variety in the implementations of the particle swarm, the most standard version uses $\alpha = 0.7298$ and $\beta = \psi/\ 2$, where $\psi = 2.9922$, following an analysis published in Clerc and Kennedy (2002). The constant α is called an *inertia weight* or *constriction coefficient*, and β is known as the *acceleration constant*.

The program evaluates the parameter vector of particle i in a function $f(\vec{x})$ and compares the result to the best result attained by i thus far, called $pbest_i$. If the current result is i's best so far, the vector $\vec{p}_i$ is updated with the current position $\vec{x}_i$, and the previous best function result $pbest_i$ is updated with the current result.

When the system is run, each particle cycles around a region centered on the centroid of the previous bests $\vec{p}_i$ and $\vec{p}_g$; as these variables are updated, the particle's trajectory shifts to new regions of the search space, the particles begin to cluster around optima, and improved function results are obtained.

The Social–Psychological Metaphor

Classical social psychology theorists considered the pursuit of *cognitive consistency* to be an important motivation for human behavior (Heider 1958; Festinger 1957; Abelson et al. 1968). Cognitive elements might have emotional or logical aspects to them which could be consistent or inconsistent with one another; several theorists identified frameworks for describing the degree of consistency and described the kinds of processes that an individual might use to increase consistency or balance, or decrease inconsistency or cognitive dissonance.

Contemporary social and cognitive psychologists frequently cast these same concepts in terms of connectionist principles. Cognitive elements are conceptualized as a network with positive and negative vertices among a set of nodes. In some models, the elements are given and the task is to reduce error by adjusting the signs and values of the connections between them, and in other models the connections are given and the goal of optimization is to find activation values that maximize coherence (Thagard 2000), harmony (Smolensky 1986), or some other measure of consistency. Typically, this optimization is performed by gradient-descent programs which psychologically model processes that are private to the individual and are perfectly rational, that is, the individual always decreases error or increases consistency among elements. The particle swarm simulates the optimization of these kinds of structures through social interaction; it is commonly observed, not only in the laboratory but in everyday life, that a person faced with a problem typically solves it by talking with other people.

A direct precursor of the particle swarm is seen in Nowak et al. (1990) cellular automaton simulation of social impact theory's predictions about interaction in human social populations. Social impact theory predicted that an individual was influenced to hold an attitude or belief in proportion to the Strength, Immediacy, and Number of sources of influence holding that position, where Strength was a measure of the persuasiveness or prestige of an individual, Immediacy was their proximity, and Number was literally the number of sources of influence holding a particular attitude or belief. In the simulation, individuals iteratively interacted, taking on the prevalent state of a binary attitude in their neighborhood, until the system reached equilibrium.

The particle swarm extends this model by supposing that various states can be evaluated, for instance, that different patterns of cognitive elements may be more or less dissonant; it assumes that individuals hold more than one attitude or belief, and that they are not necessarily binary; and Strength is replaced with a measure of self-presented success. One feature usually found in particle swarms and not in the paper by Nowak et al. is the phenomenon of persistence or momentum, the tendency of an individual to keep changing or moving in the same direction from one time-step to the next.

Thus, the particle swarm metaphorically represents the interactions of a number of individuals, none knowing what the goal is, each knowing its immediate state and its best performance in the past, each presenting its neighbors with its best success-so-far at solving a problem, each functioning as both source and target of influence in the dynamically evolving system. As individuals emulate the successes of their neighbors, the population begins to cluster in optimal regions of a search space, reliably discovering good solutions to difficult problems featuring, for instance, nonlinearity, high dimension, deceptive gradients, local optima, etc.

The Population Topology

Several kinds of topologies have been most widely used in particle swarm research; the topic is a current focus of much research. In the *gbest* topology, the population is conceptually fully connected; every particle is linked to every other. In practice, with the best neighbor canonical version, this is simpler to implement than it sounds, as it only means that every particle receives influence from the best performing member of the population.

The *lbest* topology of degree K_i comprises a ring lattice, with the particle linked to its K_i nearest neighbors on both sides in the wrapped population array.

Another structure commonly used in particle swarm research is the von Neumann or "square" topology. In this arrangement, the population is laid out in rows and columns, and each individual is connected to the neighbors above, below, and on each side of it in the toroidally wrapped population. Numerous other topologies have been used, including random (Suganthan 1999), hierarchical (Janson and Middendorf 2005), and adaptive ones (Clerc 2006).

The most important effect of the population topology is to control the spread of proposed problem solutions through the population. As a particle finds a good region of the search space, it may become the best neighbor to one of the particles it is connected to. That particle then will tend to explore in the vicinity of the first particle's success, and may eventually find a good solution there, too; it could then become the best neighbor to one of its other neighbors. In this way, information about good regions of the search space migrates through the population.

When connections are parallel, e.g., when the mean degree of particles is relatively high, then information can spread quickly through the population. On unimodal problems this may be acceptable, but where there are local optima there may be a tendency for the population to converge too soon on a suboptimal solution. The *gbest* topology has repeatedly been shown to be vulnerable to the lure of locally optimal attractors.

On the other hand, where the topology is sparse, as in the *lbest* model, problem solutions spread slowly, and subpopulations may search diverse regions of the search space in parallel. This increases the probability that the population will end up near the global optimum. It also means that convergence will be slower.

Vmax and Convergence

The particle swarm has evolved very much since it was first reported by Kennedy and Eberhart (1995) and Eberhart and Kennedy (1995). Early versions required a system constant *Vmax* to limit the velocity. Without this limit, the particles' trajectories would swing wildly out of control.

Following presentation of graphical representations of a deterministic form of the particle swarm by Kennedy (1998), early analyses by Ozcan and Mohan (1999) led to some understanding of the nature of the particle's trajectory. Analytical breakthroughs by Clerc (reported in Clerc and Kennedy (2002)), and empirical discoveries by Shi and Eberhart (1998), resulted in the application of the α constant in concert with appropriate values of the acceleration constant β. These parameters brought the particle under control, allowed convergence under appropriate conditions, and made *Vmax* unnecessary. It is still used sometimes, set to very liberal values such as a half or third of the initialization range of a variable for more efficient swarm behavior, but it is not necessary.

Step Size and Consensus

Step size in the particle swarm is inherently scaled to consensus among the particles. A particle goes in one direction on each dimension until the sign of its velocity is reversed by the accumulation of $(p - x)$ differences; then it turns around and goes the other way. As it searches back and forth, its oscillation on each dimension is centered on the mean of the previous bests $(p_{id} + p_{gd})/2$, and the standard deviation of the distribution of points that are tested is scaled to the difference between them. In fact this function is a very simple one: the standard deviation of a particle's search, when p_{id} and p_{gd} are constants, is approximately $|(p_{id} - p_{gd})|$. This means that when the particles' previous best points are far from one another in the search space, the particles will take big steps, and when they are nearer the particles will take little steps.

Over time, this usually means that exploring behavior is seen in early iterations and exploiting behavior later on as particles come to a state of consensus. If it happens, however, that a particle that has begun to converge in one part of the search space receives information about a good region somewhere else, it can return to the exploratory mode of behaving.

The Fully Informed Particle Swarm (FIPS)

Mendes (2004) reported a version of swarm that featured an alternative to the best neighbor strategy. While the canonical particle is influenced by its own previous success and the previous success of its best neighbor, the fully informed particle swarm (FIPS) allowed influence by all of a particle's neighbors. The acceleration constants were set to $\beta = \psi/\ 2$ in the traditional version; it was defined in this way because what mattered was their sum, which could be distributed among any number of difference terms. In the standard algorithm there were two of them, and thus the sum was divided by 2. In FIPS a particle of K_i degree has coefficients $\beta = \psi/K_i$.

The FIPS particle swarm removed two aspects that were considered standard features of the algorithm. First of all, the particle i no longer influenced itself directly, e.g., there is no $\vec{p}_i$ in the formula. Second, the best neighbor is now averaged in with the others; it was not necessary to compare the successes of all neighbors to find the best one.

Mendes found that the FIPS swarm was more sensitive than the canonical versions to the differences in topology. For instance, while in the standard versions the fully connected *gbest* topology meant influence by the best solution known to the entire population, in FIPS *gbest* meant that the particle was influenced by a stochastic average of the best solutions found by all members of the population; the result tended to be near-random search.

The lesson to be learned is that the *meaning* of a topology depends on the mode of interaction. Topological structure (and Mendes tested more than 1,340 of them) affects performance, but the way it affects the swarm's performance depends on how information is propagated from one particle to another.

Generalizing the Notation

Equation 2 above shows that the position is derived from the previous iteration's position plus the current iteration's velocity. By rearranging the terms, it can be shown that the current iteration's velocity $\vec{v}_i^{(t+1)}$ is the difference between the new position and the previous one: $\vec{v}_i^{(t+1)} = \vec{x}_i^{(t+1)} - \vec{x}_i^{(t)}$. Since this happened on the previous time-step as well, it can be shown that $\vec{v}_i^{(t)} = \vec{x}_i^{(t)} - \vec{x}_i^{(t-1)}$; this fact makes it possible to combine the two formulas into one:

$$x_{id}^{(t+1)} \leftarrow x_{id}^{(t)} + \alpha \left(x_{id}^{(t)} - x_{id}^{(t-1)} \right) + \sum U \left(0, \frac{\Psi}{K_i} \right) \left(p_{kd} - x_{id}^{(t)} \right) \qquad (3)$$

where K_i is the degree of node i, k is the index of i's kth neighbor, and adapting Clerc's (Clerc and Kennedy 2002) scheme $\alpha = 0.7298$ and $\psi = 2.9922$.

In the canonical best neighbor particle swarm, $K_i = 2, \forall i : i = 1, 2, \ldots, N$ and $k \in (i, g)$, that is, k takes the values of the particle's own index and its best neighbor's index. In FIPS, K_i may vary, depending on the topology, and k takes on the indexes of each of i's neighbors. Thus, Eq.3 is a generalized formula for the trajectories of the particles in the particle swarm.

This notation can be interpreted verbally as:

$$\begin{aligned} & NEW\ POSITION \\ & = CURRENT\ POSITION \\ & + PERSISTENCE \\ & + SOCIALINFLUENCE \end{aligned} \qquad (4)$$

That is, on every iteration, every particle on every dimension starts at the point it last arrived at, persists some weighted amount in the direction it was previously going, then makes some adjustments based on the differences between the best previous positions of its sources of influence and its own current position in the search space.

The Evolving Paradigm

The particle swarm paradigm is young, and investigators are still devising new ways to understand, explain, and improve the method. A divergence or bifurcation of approaches is observed: some researchers seek ways to simplify the algorithm (Peña et al. 2006; Owen and Harvey 2007), to find its essence, while others improve performance by adding features to it, e.g., Clerc (2006). The result is a rich unfolding research tradition with innovations appearing on many fronts.

Although the entire algorithm is summarized in one simple formula, it is difficult to understand how it operates or why it works. For instance, while the *Social Influence* terms point the particle in the direction of the mean of the influencers' successes, the *Persistence* term offsets that movement, causing the particle to bypass what seems to be a reasonable target. The result is a spiral-like trajectory that goes past the target and returns to pass it again, with the spiral tightening as the neighbors come to consensus on the location of the optimum.

Further, while authors often talk about the particle's velocity carrying it "toward the previous bests," in fact the velocity counterintuitively carries it *away from* the previous bests as often as toward them. It is more accurate to say the particle "explores around" the previous bests, and it is hard to describe this against-the-grain movement as "gradient descent," as some writers would like.

It is very difficult to visualize the effect of ever-changing sources of influence on a particle. A different neighbor may be best from one iteration to the next; the balance of the random numbers may favor one or another or some compromise of sources; the best neighbor could remain the same one, but may have found a better $\vec{p}_i$ since the last turn; and so on. The result is that the particle is pulled and pushed around in a complex way, with many details changing over time.

The paradoxical finding is that it is best not to give the particle information that is too good, especially early in the search trial. Premature convergence is the result of amplified consensus resulting from too much communication or overreliance on best neighbors, especially the population best. Various researchers have proposed ways to slow the convergence or clustering of particles in the search space, such as occasional reinitialization or randomization of particles, repelling forces among them, etc., and these techniques typically have the desired effect. In many cases, however, implicit methods work as well and more parsimoniously; the effect of topology on convergence rate has been mentioned here, for instance.

Binary Particle Swarms

A binary particle swarm is easily created by treating the velocity as a probability threshold (Kennedy and Eberhart 1997). Velocity vector elements are squashed in a sigmoid or other function, for instance $S(\upsilon) = 1/(1 + exp(-\upsilon))$, producing a result in (0..1). A random number is generated and compared to $S(\upsilon_{id})$ to determine whether x_{id} will be a 0 or a 1. Though discrete systems of higher cardinality have been proposed, it is difficult to define such concepts as distance and direction in a meaningful way within nominal data.

Alternative Probability Distributions

As was noted above, the particle's search is centered around the mean of the previous bests that influence it, and its variance is scaled to the differences among them. This has suggested to several researchers that perhaps the trajectory formula can be replaced, wholly or partly, by some type of random number generator that directly samples the search space in a desirable way.

Kennedy (2003) suggested simple Gaussian sampling, using a random number generator (RNG) $G(mean, s.d.)$ with the mean centered between $\vec{p}_i$ and $\vec{p}_g$, and with the standard deviation defined on each dimension as $s.d. = |(p_{id} - p_{gd})|$. This "bare bones" particle swarm eliminated the velocity component; it performed rather well on a set of test functions, but not as well as the usual version.

Krohling (2004) simply substituted the absolute values of Gaussian-distributed random numbers for the uniformly distributed values in the canonical particle swarm. He and his colleagues have had success on a range of problems using this approach. Richer and Blackwell (2006) replaced the Gaussian distribution of bare bones with a Lévy distribution. The Lévy distribution is bell-shaped like the Gaussian but with fatter tails. It has a parameter α which allows interpolation between the Cauchy distribution ($\alpha = 1$) and Gaussian ($\alpha = 2$) and can be used to control the fatness of the tails. In a series of trials, Richer and Blackwell (2006) were able to emulate the performance of a canonical particle swarm using $\alpha = 1.4$. Kennedy (2005) used a Gaussian RNG for the social influence term of the usual formula, keeping the "persistence" term found in the standard particle swarm. Variations on this format produced results that were competitive with the canonical version.

Numerous other researchers have begun exploring ways to replicate the overall behavior of the particle swarm by replacing the traditional formulas with alternative probability distributions. Such experiments help theorists understand what is essential to the swarm's behavior and how it is able to improve its performance on a test function over time.

Simulation of the canonical trajectory behavior with RNGs is a topic that is receiving a great deal of attention at this time, and it is impossible to predict where the research is leading. As numerous versions have been published showing that the trajectory formulas can be replaced by alternative strategies for selecting a series of points to sample, it becomes apparent that the essence of the paradigm is not to be found in the details of the movements of the particles, but in the nature of their interactions over time, the structure of the social network in which they are embedded, and the function landscape with which they interact, with all these factors working together gives the population the ability to find problem solutions.

Recommended Reading

Abelson RP, Aronson E, McGuire WJ, Newcomb TM, Rosenberg MJ, Tannenbaum RH (eds) (1968) Theories of cognitive consistency: a sourcebook. Rand McNally, Chicago

Clerc M (2006) Particle swarm optimization. Hermes Science Publications, London

Clerc M, Kennedy J (2002) The particle swarm: explosion, stability, and convergence in a multidimensional complex space. IEEE Trans Evol Comput 6:58–73

Eberhart RC, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceedings of the 6th international symposium on micro machine and human science, Nagoya. IEEE Service Center, Piscataway, pp 39–43

Festinger L (1957) A theory of cognitive dissonance. Stanford University Press, Stanford

Heider F (1958) The psychology of interpersonal relations. Wiley, New York

Janson S, Middendorf M (2005) A hierarchical particle swarm optimizer and its adaptive variant. IEEE Trans Syst Man Cybern Part B Cybern 35(6):1272–1282

Kennedy J (1998) The behavior of particles. In: Porto VW, Saravanan N, Waagen D, Eiben AE (eds) Evolutionary programming VII. Proceedings of the 7th annual conference on evolutionary programming, San Diego

Kennedy J (2003) Bare bones particle swarms. In: Proceedings of the IEEE swarm intelligence symposium, Indianapolis, pp 80–87

Kennedy J (2005) Dynamic-probabilistic particle swarms. In: Proceedings of the genetic and evolutionary computation conference (GECCO-2005), Washington, DC, pp 201–207

Kennedy J, Eberhart RC (1995) Particle swarm optimization. In: Proceedings of the 1995 IEEE international conference on neural networks, Perth. IEEE Service Center, Piscataway, pp 1942–1948

Kennedy J, Eberhart RC (1997) A discrete binary version of the particle swarm algorithm. In: Proceedings of the 1997 conference on systems, man, and cybernetics. IEEE Service Center, Piscataway, pp 4104–4109

Krohling RA (2004) Gaussian Swarm. A novel particle swarm optimization algorithm. Proc 2004 IEEE Conf Cybern Intell Syst 1:372–376

Mendes R (2004) Population topologies and their influence in particle swarm performance. Doctoral thesis, Escola de Engenharia, Universidade do Minho

Nowak A, Szamrej J, Latané B (1990) From private attitude to public opinion: a dynamic theory of social impact. Psychol Rev 97:362–376

Owen A, Harvey I (2007) Adapting particle swarm optimisation for fitness landscapes with neutrality. In: Proceedings of the 2007 IEEE Swarm intelligence symposium. IEEE Press, Honolulu, pp 258–265

Ozcan E, Mohan CK (1999) Particle swarm optimization: surfing the waves. In: Proceedings of the congress on evolutionary computation, Mayflower hotel, Washington, DC. IEEE Service Center, Piscataway, pp 1939–1944

Peña J, Upegui A, Eduardo Sanchez E (2006) Particle Swarm optimization with discrete recombination: an online optimizer for evolvable hardware. In: Proceedings of the 1st NASA/ESA conference on adaptive hardware and systems (AHS-2006), Istanbul. IEEE Service Center, Piscataway, pp 163–170

Richer TJ, Blackwell TM (2006) The Levy particle Swarm. In: Proceedings of the 2006 congress on evolutionary computation (CEC-2006). IEEE Service Center, Piscataway

Shi Y, Eberhart RC (1998) Parameter selection in particle Swarm optimization. In: Evolutionary programming VII: proceedings EP98. Springer, New York, pp 591–600

Smolensky P (1986) Information processing in dynamical systems: foundations of harmony theory. In: Rumelhart DE, McClelland JL, the PDP Research Group (eds) Parallel distributed processing: explorations in the microstructure of cognition, vol 1, Foundations. MIT Press, Cambridge, pp 194–281

Suganthan PN (1999) Particle Swarm optimisation with a neighbourhood operator. In: Proceedings of congress on evolutionary computation, Washington DC

Thagard P (2000) Coherence in thought and action. MIT Press, Cambridge

Partitional Clustering

Xin Jin[1] and Jiawei Han[2]
[1]PayPal Inc., San Jose, CA, USA
[2]University of Illinois at Urbana-Champaign, Urbana, IL, USA

Abstract

Partitional clustering is a type of clustering algorithms that divide a set of data points into disjoint subsets. Each data point is in exactly one subset.

Synonyms

Objective function

Definition

Partitional clustering (Han et al. 2011) decomposes a data set into a set of disjoint clusters. Given a data set of N points, a partitioning method constructs K ($N \geq K$) partitions of the data with each partition representing a cluster. That is, it classifies the data into K groups by satisfying the following requirements: (1) each group contains at least one point, and (2) each point belongs to exactly one group. For fuzzy partitioning, a point can belong to more than one group. The quality of the solution is measured by clustering criteria.

Some partitional clustering algorithms work by minimizing an objective function. For example, in K-means and K-medoids, the function (also referred as the distortion function) is

$$\sum_{i=1}^{K} \sum_{j=1}^{|C_i|} Dist(x_j, center(i)) \qquad (1)$$

where $|C_i|$ is the number of points in cluster i and $Dist(x_j, center(i))$ is the distance between point x_j and center i. Depending on the need of the applications, different distance functions can be used, such as Euclidean distance and L_1 norm.

Major Algorithms

Many algorithms can be used to perform partitional data clustering; representative technologies include K-means (Lloyd 1957), K-medoids (Kaufman and Rousseeuw 2005), quality threshold (QT) (Heyer et al. 1999), expectation-maximization (EM) (Dempster et al. 1977), mean shift (Comaniciu and Meer 2002), locality-sensitive hashing (LSH) (Gionis et al. 1999), K-way spectral clustering (Luxburg 2007), etc. In the K-means algorithm, each cluster is represented by the mean value of the points in the cluster. For the K-medoids algorithm, each cluster is represented by one of the points located near the center of the cluster. Instead of setting the cluster number K, the QT algorithm uses the maximum cluster diameter as a parameter to find clusters with guaranteed quality. Expectation-maximization clustering performs expectation-maximization analysis based on statistical modeling of the data distribution, and it has more parameters. Mean shift is a nonparameter algorithm to find any shape of clusters using density estimator. Locality-sensitive hashing-based method performs clustering by hashing similar points to the same bin. K-way spectral clustering algorithm represents the data as a graph and performs graph partitioning to find clusters.

Cross-References

- ▶ K-Means Clustering
- ▶ K-Medoids Clustering
- ▶ K-Way Spectral Clustering
- ▶ Quality Threshold Clustering

Recommended Reading

Comaniciu D, Meer P (2002) Mean shift: a robust approach toward feature space analysis. IEEE Trans Pattern Anal Mach Intell 24(5):603–619

Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J R Stat Soc Ser B (Methodol) 39(1):1–38

Gionis A, Indyk P, Motwani R (1999) Similarity search in high dimensions via hashing. In: proceedings of the 25th international conference on very large data

bases (VLDB'99), San Francisco. Morgan Kaufmann Publishers Inc, pp 518–529
Han J, Kamber M, Pei J (2011) Data mining: concepts and techniques, 3rd edn. Morgan Kaufmann, San Francisco
Heyer L, Kruglyak S, Yooseph S (1999) Exploring expression data: identification and analysis of coexpressed genes. Genome Res 9:1106–1115
Kaufman L, Rousseeuw PJ (2005) Finding groups in data: an introduction to cluster analysis. Wiley series in probability and statistics. Wiley-Interscience, Hoboken
Lloyd SP (1957) Least squares quantization in PCM. Technical report RR-5497, Bell Lab
Luxburg U (2007) A tutorial on spectral clustering. Stat Comput 17(4):395–416

Passive Learning

A ▸ passive learning system plays no role in the selection of its ▸ training data. Passive learning stands in contrast to ▸ active learning.

PCA

▸ Principal Component Analysis

PCFG

▸ Probabilistic Context-Free Grammars

Phase Transitions in Machine Learning

Lorenza Saitta[1] and Michele Sebag[2]
[1]Università del Piemonte Orientale, Alessandria, Italy
[2]CNRS – INRIA – Université Paris-Sud, Orsay, France

Synonyms

Statistical physics of learning; Threshold phenomena in learning; Typical complexity of learning

Definition

Phase transition (PT) is a term originally used in physics to denote a sudden transformation of a system from one state to another, such as from liquid to solid or to gas state (phase). It is used, by extension, to describe any abrupt change in one of the *order* parameters describing an arbitrary system, when a *control* parameter approaches a *critical* value.

Far from being limited to physical systems, PTs are ubiquitous in sciences, notably including computational science. Typically, hard combinatorial problems display a PT with regard to the probability of existence of a solution. Note that the notion of PT cannot be studied in relation to single-problem instances: it refers to emergent phenomena in an *ensemble* of problem instances, governed by a given probability distribution.

Motivation and Background

Cheeseman et al. (1991) were most influential in starting the study of PTs in artificial intelligence, experimentally showing the presence of a PT containing the most difficult instances for various **NP**-complete problems. Since then, the literature flourished both in breadth and depth, witnessing an increasing transfer of knowledge and results between statistical physics and combinatorics.

As far as machine learning (ML) can be formulated as a combinatorial optimization problem (Mitchell 1982), it is no surprise that PTs emerge in many of its facets. Early results have been obtained in the field of relational learning, either logic (Botta et al. 2003; Giordana and Saitta 2000) or kernel (Gaudel et al. 2008) based. PTs have been studied in neural networks (Demongeot and Sené 2008; Engel and Van den Broeck 2001), grammatical inference (Cornuéjols and Sebag 2008), propositional classification (Baskiotis and Sebag 2004; Rückert and De Raedt 2008), and sparse regression (Donoho and Tanner 2005).

Two main streams of research emerge from the study of PT in computational problems. On the one hand, locating the PT enables very difficult problem instances to be generated, those which are most relevant to benchmarks and comparative assessment of new algorithms. On the other hand, PT studies stimulate the analysis of algorithmic *typical case complexity*, as opposed to the standard worst-case analysis of algorithmic complexity. It is well known that while many algorithms require exponential resources in the worst case, they are effective for a vast majority of problem instances. Studying their typical runtime thus makes sense in a probabilistic perspective. The typical runtime not only reflects the most probable runtime; overall, the probability of deviating from this typical complexity goes to zero as the problem size increases.

Relational Learning

In a seminal paper, Mitchell characterized ML as a search problem (Mitchell 1982). Much attention has ever since been devoted to every component of a search problem: the search space, the search goal, and the search engine.

The search space $\mathcal{H}$ reflects the language $\mathcal{L}$ chosen to express the target knowledge, termed ▶ *hypothesis language*. The reader is referred to other entries of the encyclopedia (▶ *Attribute-value* representation, ▶ *First-order logic*, ▶ *Relational learning*, and ▶ *Inductive Logic Programming*) for a comprehensive presentation of the hypothesis languages and related learning approaches.

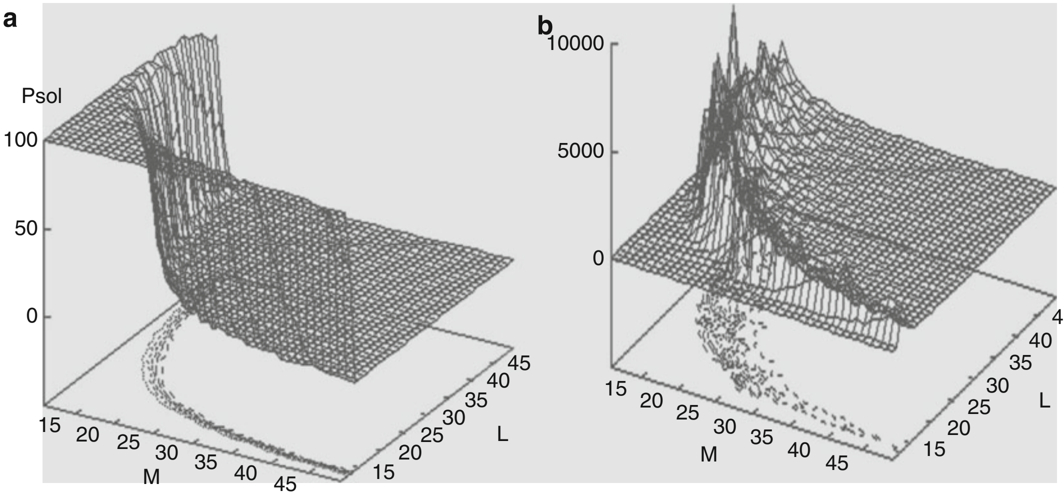
Typically, a learner proceeds iteratively: given a set $\mathcal{E}$ of examples labeled after a target concept ω, the learner maintains a list of candidate hypotheses, assessing their *completeness* (the proportion of positive examples they cover) and their *consistency* (the proportion of negative examples they do not cover) using a ▶ *covering test*. The covering test, checking whether some hypothesis h covers some example e, is thus a key component of the learning process, launched a few hundred thousand times in each learning run on medium-size problems.

While in propositional learning the covering test is straightforward and computationally efficient, in first-order logics, one must distinguish between *learning from interpretation* (h covers a set of facts e iff e is a model for h) and *learning from entailment* (h covers a clause e iff h entails e) (De Raedt 1997). A correct, but incomplete covering test, the ▶ θ-subsumption test defined by Plotkin (1970), is most often used for its decidability properties, and much attention has been paid to optimizing it (Maloberti and Sebag 2004).

As shown by Giordana and Saitta (2000), the θ-subsumption test is equivalent to a constraint satisfaction problem (CSP). A finite CSP is a tuple ($\mathbf{X}$, $\mathbf{R}$, D), where $\mathbf{X} = \{x_1, \ldots x_n\}$ is a set of variables, $\mathbf{R} = \{R_1, \ldots R_c\}$ is a set of constraints (relations), and D is the variable domain. Each relation R_h involves a subset of variables $x_{i_1}, \ldots, x_{i_k}$ in $\mathbf{X}$; it specifies all tuples of values $(a_{i_1}, \ldots, a_{i_k})$ in D^k such that the assignment $([x_{i_1} = a_{i_1}] \wedge \ldots \wedge [x_{i_k} = a_{i_k}])$ satisfies R_h. A CSP is satisfiable if there exists a tuple $(a_1, \ldots, a_n) \in D^n$ such that the assignment $([x_i = a_i], i = 1, \ldots, n)$ satisfies all relations in $\mathbf{R}$. Solving a CSP amounts to finding such a tuple (solution) or showing that none exists.

The probability for a random CSP instance to be satisfiable shows a PT with respect to the constraint density (control parameter $p_1 = \frac{2c}{n(n-1)}$) and constraint tightness ($p_2 = 1 - \frac{N}{L^2}$), where N denotes the cardinality of each constraint (assumed to be equal for all constraints) and L is the number of constants in the example (the universe).

The relational covering test being a CSP, a PT was expected and has been confirmed by empirical evidence (Botta et al. 1999; Giordana and Saitta 2000). The order parameter is the probability of hypothesis h to cover example e; the control parameters are the number m of predicates and the number n of variables in h, on the one hand, and the number N of literals built on each predicate symbol (relation) and the number L of constants in the example e, on the other hand. As shown in Fig. 1a, the covering probability is close to 1 (YES region) when h is general comparatively to e; it abruptly decreases to 0 (NO region) as the number m of predicates in h increases and/or the number L of constants in e decreases. In the PT region, a high peak



Phase Transitions in Machine Learning, Fig. 1 PT of the covering test (h, e) versus the number m of predicates in h and the number L of constants in e. The number n of variables is set to 10, and the number N of literals per predicate is set to 100. (**a**) Percentage of times the covering test succeeds. (**b**) Runtime of the covering test, averaged over 100 pairs (h, e) independently generated for each pair (m, L)

of empirical complexity of the covering test is observed (Fig. 1b).

The PT of the covering test has deep and far-reaching effects on relational learning. By definition, nontrivial hypotheses (covering some examples but not all) mostly belong to the PT region. The learner, searching for hypotheses covering the positive and rejecting the negative examples, must explore this region and thus cannot avoid the associated computational cost. More generally, the PT region acts as an *attractor* for any learner aimed at complete and consistent hypotheses.

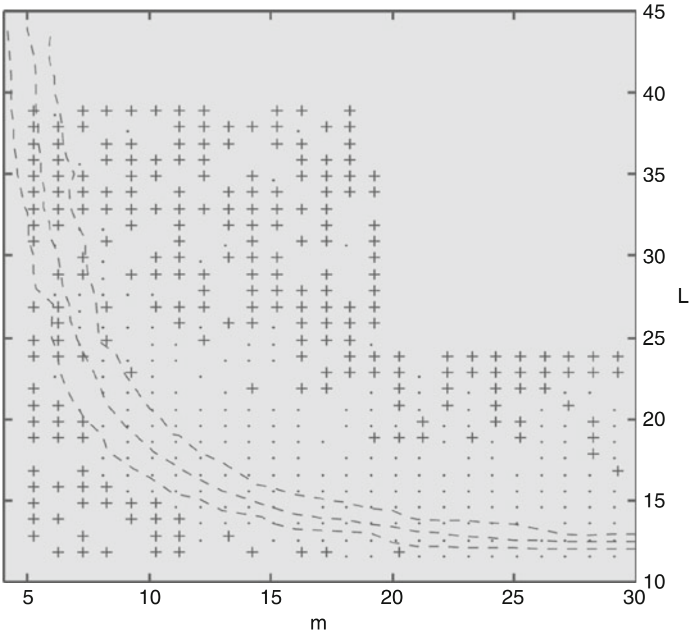
Secondly, top-down learners must traverse the plateau of overly general hypotheses (YES region) before arriving at the PT region. In the YES region, as all hypotheses cover most examples, the learner does not have enough information to make relevant choices; the chance of gradually arriving at an accurate description of the target concept thus becomes very low. Actually, a *blind spot* has been identified close to the PT (Botta et al. 2003): when the target concept lies in this region (relatively to the available examples), every state-of-the-art top-down relational learner tends to build random hypotheses, that is, the learned hypotheses behave like random guessing on the test set (Fig. 2).

This negative result has prompted the design of new relational learners aimed at learning in the PT region and using either prior knowledge about the size of the target concept (Ales Bianchetti et al. 2002) or near-miss examples (Alphonse and Osmani 2008).

Relational Kernels and MIL Problems

Relational learning has been revisited through the so-called kernel trick (Cortes and Vapnik 1995), first pioneered in the context of ▶ Support Vector Machines. Relational kernels, inspired from Haussler's convolutional kernels (Haussler 1999), have been developed for, e.g., strings, trees, or graphs. For instance, $K(\mathbf{x}, \mathbf{x}')$ might count the number of patterns shared by relational structures $\mathbf{x}$ and $\mathbf{x}'$. Relational kernels thus achieve a particular type of ▶ propositionalization (Kramer et al. 2001), mapping every relational example onto a propositional space defined after the training examples.

The question of whether relational kernels enable to avoid the PT faced by relational learning, described in the previous section, was investigated by Gaudel et al. (2007), focusing on the so-called ▶ multi-instance learning



Phase Transitions in Machine Learning, Fig. 2 Competence map of FOIL versus number m of predicates in the target concept and number L of constants in the examples. The target concept involves $n = 4$ variables and each example contains $N = 100$ literals built on each predicate symbol. For each pair (m, L), a target concept ω has been generated independently, balanced 200-example training, and test sets have been generated and labeled after ω. FOIL has been launched on the training set, and the predictive accuracy of the hypothesis has been assessed on the test set. Symbol “—” indicates a predictive accuracy greater than 90 %; symbol “—” indicates a predictive accuracy close to 50 % (akin random guessing)

(MIL) setting. The MIL setting, pioneered by Dietterich et al. (1997), is considered to be the “missing link” between relational and propositional learning (De Raedt 1998).

Multi-instance Learning: Background and Kernels

Formally, an MI example $\mathbf{x}$ is a bag of (propositional) instances noted $x^{(1)}$, …, $x^{(N)}$, where $x^{(j)} \in \mathbf{R}^d$. In the original MI setting (Dietterich et al. 1997), an example is labeled positive iff it includes at least one instance satisfying some target concept C:

$$pos(\mathbf{x}) \text{ iff } \exists\, i \in 1 \ldots N \; s.t.\; C(x^{(i)}).$$

More generally, in application domains such as image categorization, the example label might depend on the properties of several instances:

$$pos(\mathbf{x}) \text{ iff } \forall\, j = 1 \ldots m,\ \exists\, i_j \in 1 \ldots N \; s.t.\; C_j (x^{(i_j)}).$$

In this more general setting, referred to as *presence-based* setting, it has been shown that MIL kernels also have a PT (Gaudel et al. 2007).

Let us consider bag kernels K, built on the top of propositional kernels k on $\mathbf{R}^d$ as follows:

$$K(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}).f(\mathbf{x}') \sum_{k=1}^{N} \sum_{\ell=1}^{N'} k(x^{(k)}, x'^{(\ell)}) \tag{1}$$

where $\mathbf{x} = (x^{(1)}, \ldots, x^{(N)})$ and $\mathbf{x}' = (x'^{(1)}, \ldots, x'^{(N')})$ denote two MI examples and $f(\mathbf{x})$ corresponds to a normalization term, e.g., $f(\mathbf{x}) = 1$ or $1/N$ or $1/\sqrt{K(\mathbf{x}, \mathbf{x})}$.

By construction, such MI-kernels thus consider the average similarity among the example instances, while relational learning is usually concerned with finding existential concepts.

The MI-SVM PT

After Botta et al. (2003) and Giordana and Saitta (2000), the competence of MI-kernels was experimentally assessed using artificial problems. Each problem involves m sub-concept s C_i: a given sub-concept corresponds to a region of the d-dimensional space, and it is satisfied by an MI example **x** if at least one instance in **x** belongs to this region. An instance is said to be relevant if it belongs to some C_i region.

Let n (respectively n') denote the number of relevant instances in positive (respectively negative) examples. Let further τ denote the number of sub-concept s not satisfied by negative examples (by definition, a positive example satisfies all sub-concept s).

Empirical investigations (Gaudel et al. 2007) show that:

- The $n = n'$ region is a failure region, where hypotheses learned by relational MI-SVMs do no better than random guessing (Fig. 3). In other words, while MI-SVMs grasp the notion of relevant instances, they still fail in the "truly relational region" where positive and negative examples only differ in the distribution of the relevant instances.
- The width of the failure region increases as τ increases, i.e., when fewer sub-concept s are satisfied by negative examples. This unexpected result is explained from the variance of the kernel-based propositionalization: the larger τ, the more the distribution of the positive and negative propositionalized examples overlap, hindering the discrimination.

Propositional Learning and Sparse Coding

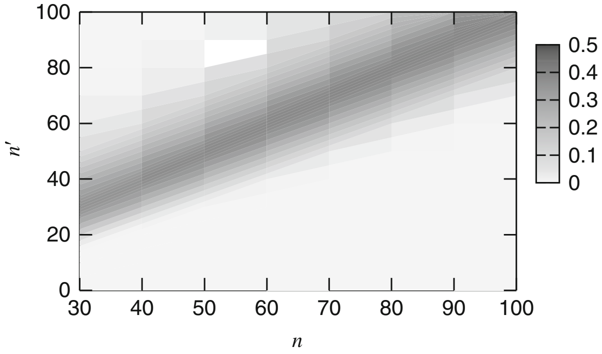
Interestingly, the emergence of a PT is not limited to relational learning. In the case of (context-free) *grammar induction*, for instance (Cornuéjols and Sebag 2008), the coverage of the candidate grammar was found to abruptly go to 1 along (uniform) generalization, as depicted in Fig. 4.

Propositional learning also displays some PTs both in the classification (Baskiotis and Sebag 2004; Rückert and De Raedt 2008) and in the regression (Cands 2008; Donoho and Tanner 2005) context.

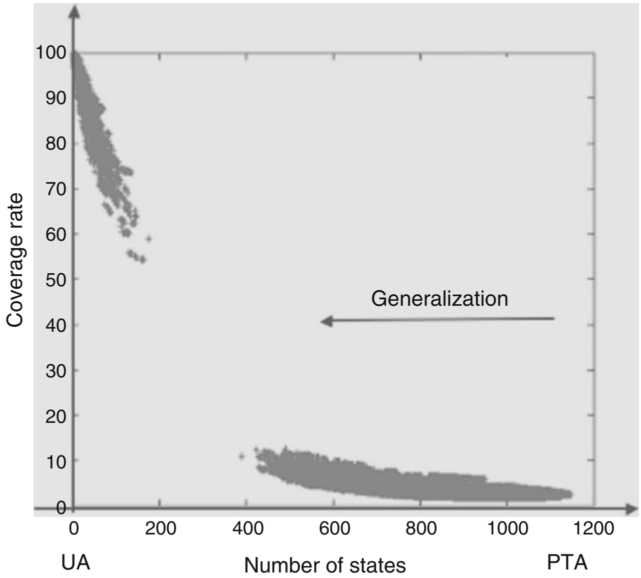
Propositional Classification

Given a target hypothesis language, classification in discrete domains most often aims at the simplest expression complying with the training examples.

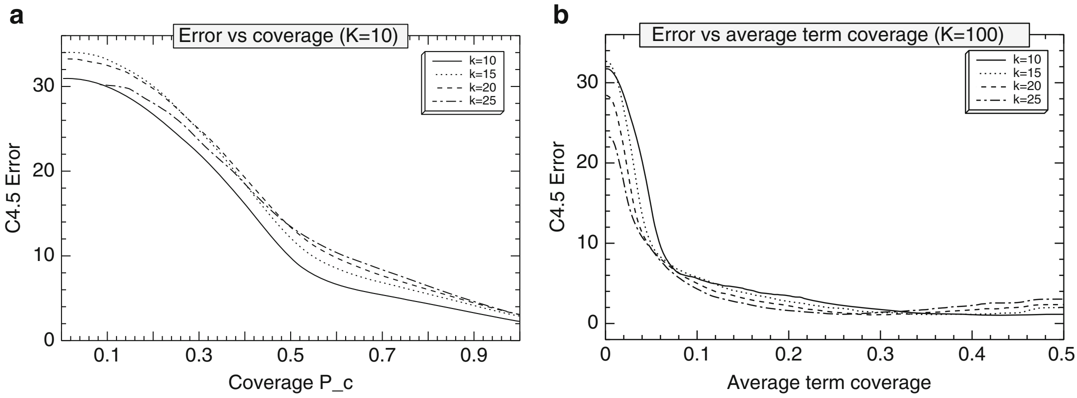
Considering randomly generated positive and negative examples, Rückert and De Raedt (2008) investigated the existence of k-term DNF solutions (disjunction of at most k conjunctions of literals) and showed that the probability of solution abruptly drops as the number of negative examples increases. They proposed a combinatorial optimization algorithm to find a k-term DNF



Phase Transitions in Machine Learning, Fig. 3 MI-SVM failure region in the (n, n') plane. Each (n, n') point reports the test error, averaged on 40 artificial problems



Phase Transitions in Machine Learning, Fig. 4 Gap emerging during learning in the relationship between the number of nodes of the inferred grammar and the coverage rate



Phase Transitions in Machine Learning, Fig. 5 C4.5 error versus concept coverage (**a**) and average term coverage (**b**) in k-term DNF languages. The reported curve is obtained by Gaussian convolution with empirical data (15,000 learning problems, each one involving a 800-example dataset)

complying with the training examples except at most $\varepsilon\,\%$ of them (Rückert and De Raedt 2008).

Considering positive and negative examples generated after some k-term DNF target concept ω, Baskiotis and Sebag examined the solutions built by C4.5 Rules (Quinlan 1993), among the oldest and still most used discrete learning algorithms. The observed variable is the generalization error on a test set; the order variables are the coverage of ω and the average coverage of the conjuncts in ω. Interestingly, C4.5 displays a PT behavior (Fig. 5): the error abruptly increases as the coverage and average coverage decrease.

Propositional Regression

▶ Linear regression aims at expressing the target variable as the weighted sum of the N descriptive variables according to some vector $\mathbf{w}$. When the number N of variables is larger than the number n of examples, one is interested in finding the most sparse $\mathbf{w}$ complying with the training examples (s.t. $< \mathbf{w}, \mathbf{x_i} >= \mathbf{y_i}$). The sparsity criterion consists of minimizing the L_0 norm of

w (number of nonzero coefficients in **w**), which defines an NP optimization problem. A more tractable formulation is obtained by minimizing the L_1 norm instead:

$$\text{Find } \arg\min_{\mathbf{w}\in\mathrm{R}^N}\{||\mathbf{w}||_1 \text{ subject to } < \mathbf{w}, \mathbf{x_i} > = \mathbf{y_i}, i = 1\ldots n\}. \tag{2}$$

A major result in the field of sparse coding can be stated as: *Let* $\boldsymbol{w}^*$ *be the solution of Eq.* (2)*; if it is sufficiently sparse,* $\boldsymbol{w}^*$ *also is the most sparse vector subject to* $< \boldsymbol{w}, \boldsymbol{x_i} > = \boldsymbol{y_i}$ (Donoho and Tanner 2005). In such cases, the L_0 norm minimization can be solved by L_1 norm minimization (an NP optimization problem is solved using linear programming). More generally, the equivalence between L_0 and L_1 norm minimization shows a PT behavior: when the sparsity of the solution is lower than a given threshold w.r.t the problem size (lower curve in Fig. 6), the NP/LP equivalence holds strictly; further, there exists a region (between the upper and lower curves in Fig. 6) where the NP/LP equivalence holds with high probability.

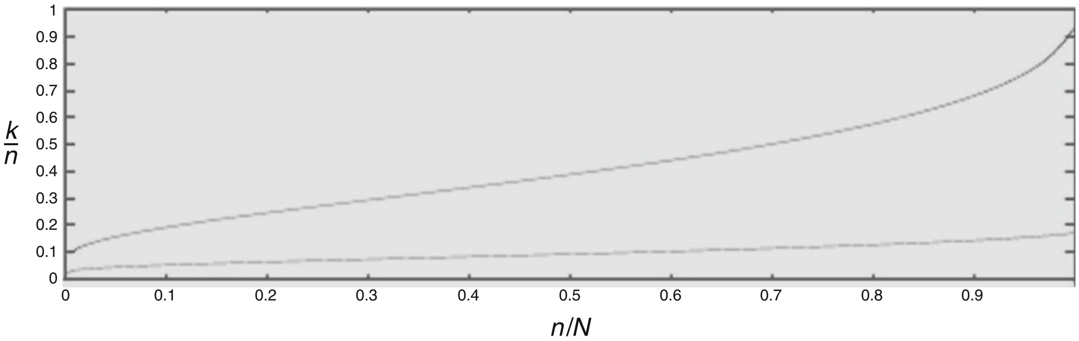
This highly influential result bridges the gap between the statistical and algorithmic objectives. On the statistical side, the importance of sparsity in terms of robust coding (hence learning) is acknowledged since the beginnings of information theory; on the algorithmic side, the sparsity criterion cannot be directly tackled as it boils down to solving a combinatorial optimization problem (minimizing a L_0 norm). The above result reconciles sparsity and tractability by noting that under some conditions, the solution of the L_0 minimization problem can be found by solving the (tractable) L_1 minimization problem: whenever the solution of the latter problem is "sufficiently" sparse, it is also the solution of the former problem.

Perspectives

Since the main two formulations of ML involve constraint satisfaction and constrained optimization, it is no surprise that CSP PTs manifest themselves in ML. The diversity of these manifestations, ranging from relational learning (Botta et al. 2003) to sparse regression (Donoho and Tanner 2005), has been illustrated in this entry, without pretending exhaustivity.

Along this line, the research agenda and methodology of ML can benefit from the lessons learned in the CSP field. Firstly, algorithms must be assessed on problems lying in the PT region; results obtained on problems in the easy regions are likely to be irrelevant (*playing in the sandbox* Hogg et al. 1996).

In order to do so, the PT should be localized through defining control and order parameters, thus delineating several regions in the control parameter space (ML landscape). These regions expectedly correspond to different types of ML difficulty, beyond the classical computational complexity perspective.



Phase Transitions in Machine Learning, Fig. 6 Strong and weak PT in sparse regression (Donoho and Tanner 2005). The x-axis is the ratio between the number n of constraints and the number N of variables; the y-axis is the ratio between the number k of variables involved in the solution and n

Secondly, the response of a given algorithm to these difficulties can be made available through a competence map, depicting its average performance conditionally to the value of the control parameters as shown in Figs. 2 and 3.

Finally, such competence maps can be used to determine whether a given algorithm is a priori relevant in a given region of the control parameter space and support the algorithm selection task (a.k.a. meta-learning; see, e.g., http://www.cs.bris.ac.uk/Research/MachineLearning/metal.html).

Recently, phase transitions have also emerged in learning more complex structures, such as complex networks. For instance, Xhang et al. (Zhang et al. 2014), following previous work, investigated the transitions occurring in the possibility of discovering communities in sparse networks using a semisupervised clustering approach. In their approach, the control parameter is the fraction α of nodes in the network, whose label is known, and they found both a first-order and a second-order phase transition.

Recommended Reading

Ales Bianchetti J, Rouveirol C, Sebag M (2002) Constraint-based learning of long relational concepts. In: Sammut C (ed) Proceedings of international conference on machine learning, ICML'02. Morgan Kauffman, San Francisco, pp 35–42

Alphonse E, Osmani A (2008) On the connection between the phase transition of the covering test and the learning success rate. Mach Learn 70(2–3):135–150

Baskiotis N, Sebag M (2004) C4.5 competence map: a phase transition-inspired approach. In: Proceedings of international conference on machine learning. Morgan Kaufman, Banff, pp 73–80

Botta M, Giordana A, Saitta L (1999) An experimental study of phase transitions in matching. In: Proceedings of the 16th international joint conference on artificial intelligence, Stockholm, pp 1198–1203

Botta M, Giordana A, Saitta L, Sebag M (2003) Relational learning as search in a critical region. J Mach Learn Res 4:431–463

Cands EJ (2008) The restricted isometry property and its implications for compressed sensing. Compte Rendus de l'Academie des Sciences, Paris, Serie I 346:589–592

Cheeseman P, Kanefsky B, Taylor W (1991) Where the really hard problems are. In: Myopoulos R, Reiter J (eds) Proceedings of the 12th international joint conference on artificial intelligence, Sydney. Morgan Kaufmann, San Francisco, pp 331–340

Cornuéjols A, Sebag M (2008) A note on phase transitions and computational pitfalls of learning from sequences. J Intell Inf Syst 31(2):177–189

Cortes C, Vapnik VN (1995) Support-vector networks. Mach Learn 20:273–297

De Raedt L (1997) Logical setting for concept-learning. Artif Intell 95:187–202

De Raedt L (1998) Attribute-value learning versus inductive logic programming: the missing links. In: Proceedings inductive logic programming, ILP. LNCS, vol 2446. Springer, London, pp 1–8

Demongeot J, Sené S (2008) Boundary conditions and phase transitions in neural networks. Simulation results. Neural Netw 21(7):962–970

Dietterich T, Lathrop R, Lozano-Perez T (1997) Solving the multiple-instance problem with axis-parallel rectangles. Artif Intell 89(1–2):31–71

Donoho DL, Tanner J (2005) Sparse nonnegative solution of underdetermined linear equations by linear programming. Proc Natl Acad Sci 102(27):9446–9451

Engel A, Van den Broeck C (2001) Statistical mechanics of learning. Cambridge University Press, Cambridge

Gaudel R, Sebag M, Cornuéjols A (2007) A phase transition-based perspective on multiple instance kernels. In: Proceedings of international conference on inductive logic programming, ILP, Corvallis, pp 112–121

Gaudel R, Sebag M, Cornuéjols A (2008) A phase transition-based perspective on multiple instance kernels. Lect Notes Comput Sci 4894:112–121

Giordana A, Saitta L (2000) Phase transitions in relational learning. Mach Learn 41(2):17–251

Haussler D (1999) Convolutional kernels on discrete structures. Technical report, Computer Science Department, University of California at Santa Cruz

Hogg T, Huberman BA, Williams CP (eds) (1996) Artificial intelligence: special issue on frontiers in problem solving: phase transitions and complexity, vol 81(1–2). Elsevier

Kramer S, Lavrac N, Flach P (2001) Propositionalization approaches to relational data mining. In: Dzeroski S, Lavrac N (eds) Relational data mining. Springer, New York, pp 262–291

Maloberti J, Sebag M (2004) Fast theta-subsumption with constraint satisfaction algorithms. Mach Learn J 55:137–174

Mitchell TM (1982) Generalization as search. Artif Intell 18:203–226

Plotkin G (1970) A note on inductive generalization. In: Machine intelligence, vol 5. Edinburgh University Press, Edinburgh

Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, San Francisco

Rückert U, De Raedt L (2008) An experimental evaluation of simplicity in rule learning. Artif Intell 172(1):19–28
Zhang P, Moore C, Zdeborova L (2014) Phase transitions in semisupervised clustering of sparse networks. CoRR vol abs/1404.7789

Piecewise Constant Models

▶ Regression Trees

Piecewise Linear Models

▶ Model Trees

Plan Recognition

▶ Inverse Reinforcement Learning

Polarity Learning on a Stream

▶ Opinion Stream Mining

Policy Gradient Methods

Jan Peters[1,2,4] and J. Andrew Bagnell[3]
[1]Department of Empirical Inference,
Max-Planck Institute for Intelligent Systems,
Tübingen, Germany
[2]Intelligent Autonomous Systems, Computer
Science Department, Technische Universität
Darmstadt, Darmstadt, Hessen, Germany
[3]Carnegie Mellon University, Pittsburgh, PA,
USA
[4]Max Planck Institute for Biological
Cybernetics, Tübingen, Germany

Abstract

Already Richard Bellman suggested that searching in policy space is fundamentally different from value function-based reinforcement learning — and frequently advantageous, especially in robotics and other systems with continuous actions. Policy gradient methods optimize in policy space by maximizing the expected reward using a direct gradient ascent. We discuss their basics and the most prominent approaches to policy gradient estimation.

Definition

A policy gradient method is a ▶ reinforcement learning approach that directly optimizes a parametrized control policy by a variant of gradient descent. These methods belong to the class of ▶ policy search techniques that maximize the expected return of a policy from a fixed class, in contrast with ▶ value function approximation approaches that derive policies indirectly from an estimated value function. Policy gradient approaches have various advantages: they enable the straightforward incorporation of domain knowledge in policy parametrization; often an optimal policy is more compactly represented than the corresponding value function; many such methods guarantee to convergence to at least a locally optimal policy; the methods naturally handle continuous states and actions and often even imperfect state information. The countervailing drawbacks include difficulties in off-policy settings, the potential for very slow convergence and high sample complexity, as well as identifying local optima that are not globally optimal.

Structure of the Learning System

Policy gradient methods center around a parametrized policy π_θ, also known as a ▶ direct controller, with parameters θ that defines the selection of actions a given the state s. Such a policy may either be deterministic $a = \pi_\theta(s)$ or stochastic $a \sim \pi_\theta(a|s)$. This choice also affects the class of policy gradient algorithms applicable (stochastic policies often lead to smooth differentiable objective with gradients that can be estimated via likelihood ratio methods (Williams 1992), where a deterministic policy may lead to a non-smooth optimization problem), influences how the exploration-exploitation dilemma is addressed (e.g., a stochastic policy naturally chooses novel actions while a deterministic policy requires the perturbation of policy parameters or sufficient stochasticity in the system to achieve exploration), and may affect the quality of optimal solution (e.g., for a time-invariant or stationary policy, the optimal policy can be stochastic Sutton et al. 2000). Frequently used policy classes include Gibbs distributions $\pi_\theta(a|s) = \exp(\phi(s,a)^T\theta)/\sum_b \exp(\phi(s,b)^T\theta)$ for discrete problems (Sutton et al. 2000; Bagnell 2004) and, for continuous problems, Gaussian policies $\pi_\theta(a|s) = \mathcal{N}(\phi(s,a)^T\theta_1, \theta_2)$ with an exploration parameter θ_2 (Williams 1992; Peters and Schaal 2008).

Expected Return

Policy gradient methods seek to optimize the expected return of a policy π_θ,

$$J(\theta) = Z_\gamma E\left\{\sum_{k=0}^{H} \gamma^k r_k\right\},$$

where $\gamma \in [0,1]$ denotes a discount factor, a normalization constant Z_γ, and H the planning horizon. For finite H, we have an episodic reinforcement learning scenario where the truly optimal policy is nonstationary and the normalization does not matter. For an infinite horizon $H = \infty$, we choose the normalization to be $Z_\gamma \equiv (1-\gamma)$ for $\gamma < 1$ and $Z_1 \equiv \lim_{\gamma\to 1}(1-\gamma) = 1/H$ for ▶ average reward reinforcement learning problem where $\gamma = 1$.

Gradient Descent in Policy Space

Policy gradient methods follow an estimate the gradient of the expected return

$$\theta_{k+1} = \theta_k + \alpha_k g(\theta_k),$$

where $g(\theta_k) \approx \nabla_\theta J(\theta)|_{\theta=\theta_k}$ is a gradient estimate for the policy with parameters $\theta = \theta_k$ after update k (with an initial policy θ_0) and α_k denotes a learning rate. If the gradient estimator is unbiased, $\sum_{k=0}^{\infty} \alpha_k \to \infty$ while $\sum_{k=0}^{\infty} \alpha_k^2$ remains bounded, convergence to a local minimum can be guaranteed. In optimal control, model-based gradient methods have been used for optimizing policies since the 1960s (Pontryagin et al. 1962). While these are used machine learning community (e.g., differential dynamic programming with learned models), they may be numerically brittle and must rely on accurate, deterministic models. Hence, they may suffer significantly from optimization biases (i.e., if possible, they will reach a higher average return on the approximate model than possible on the real system by exploiting the shortcomings of the model) and are not generally applicable as learning problems often include discrete elements and maybe very difficult to learn effective predictive models.

Several model-free alternatives can be found in the simulation-based optimization literature (Fu 2006), including, e.g., finite-difference gradients, likelihood ratio approaches, response-surface methods, and mean-valued, weak derivatives. The advantages and disadvantages of these different approaches remain a fiercely debated topic (Fu 2006). In machine learning, the first two approaches have largely dominated gradient-based approaches to ▶ policy search, although response surface methods are arriving especially in the context of Bayesian optimization for policy search.

Finite Difference Gradients

The simplest policy gradient approaches with perhaps the most practical applications (see Bagnell (2004) and Peters and Schaal (2008) for robotics application of this method) estimate the gradient by perturbing the policy parameters. For a current policy θ_k with expected return $J(\theta_k)$, this approach will create perturbed policies $\hat{\theta}_i = \theta_k + \delta\theta_i$ with the approximated expected returns given by $J(\hat{\theta}_i) \approx J(\theta_k) + \delta\theta_i^T g$ where $g = \nabla_\theta J(\pi_\theta)|_{\theta=\theta_k}$. Such returns are typically estimated by simulation. The gradient can then be estimated by linear regression; i.e., we obtain

$$g = (\Delta\Theta^T\Delta\Theta)^{-1}\Delta\Theta^T\Delta J,$$

with parameter perturbations $\Delta\Theta = [\delta\theta_1, \ldots, \delta\theta_n]$ and mean-subtracted roll-out returns $\delta J_n = J(\hat{\theta}_i) - \overline{J(\theta_k)}$ form $\Delta J = [\delta J_1, \ldots, \delta J_n]$. The choice of the parameter perturbation largely determines the performance of the approach (Spall 2003). Limitations particular to this approach include the need for many exploratory samples, the sensitivity of the system with respect to each parameter may differs by orders of magnitude, small changes in a single parameter may render a system unstable, and stochasticity requires particular care in optimization (e.g., multiple samples, fixed random seeds, etc.), see Glynn (1990) and Spall (2003). This method is additionally referred to as the *naive Monte-Carlo policy gradient.*

Likelihood Ratio Gradients

The likelihood ratio method relies upon the stochasticity of either the policy for model-free approaches, or the system in the model-based case. Hence, it requires no explicit parameter exploration and may cope better with noise as well as parameter perturbation sensitivity problems. Moreover, in the model-free setting, in contrast with naive Monte-Carlo estimation, it potentially benefits from more assumptions on the policy parameterization. Denoting a time-indexed sequence of states, actions, and rewards of the joint system composed of the policy and environment as a *path*, a parameter setting induces a path distribution $p_\theta(\tau)$ and rewards $R(\tau) = Z_\gamma \sum_{k=0}^{H} \gamma^k r_k$ along a path τ. Thus, we may write the gradient of the expected return as

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \int p_\theta(\tau) R(\tau) d\tau \\ &= \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) R(\tau) d\tau \\ &= E\{\nabla_\theta \log p_\theta(\tau) R(\tau)\}.\end{aligned}$$

If our system $p(s'|s,a)$ is Markovian, we may use $p_\theta(\tau) = p(s_0) \prod_{h=0}^{H} p(s_{k+1}|s_k, a_k) \pi_\theta(a_k|s_k)$ for a stochastic policy $a \sim \pi_\theta(a|s)$ to obtain the model-free policy gradient estimator known as Episodic REINFORCE (Williams 1992)

$$\nabla_\theta J(\theta) = Z_\gamma E\left\{\sum_{h=0}^{H} \gamma^k \nabla_\theta \log \pi_\theta(a_k|s_k) \sum_{k=h}^{H} \gamma^{k-h} r_k\right\},$$

and for the deterministic policy $a = \pi_\theta(s)$, the model-based policy gradient

$$\nabla_\theta J(\theta) = Z_\gamma E\left\{\sum_{h=0}^{H} \gamma^k \left(\nabla_a \log p(s_{k+1}|s_k, a_k)^T \nabla_\theta \pi_\theta(s)\right) \sum_{k=h}^{H} \gamma^{k-h} r_k\right\},$$

follows from $p_\theta(\tau) = p(s_0) \prod_{h=0}^{H} p(s_{k+1}|s_k, \pi_\theta(s_k))$.

Note that all rewards preceding an action may be omitted as the cancel out in expectation. Using a state-action value function $Q^{\pi_\theta}(s,a,h) = E\left\{\sum_{k=h}^{H} \gamma^{k-h} r_k \middle| s, a, \pi_\theta\right\}$ (see ▶ value function approximation), we can rewrite REINFORCE in its modern form

$$\nabla_\theta J(\theta) = Z_\gamma E\left\{\sum_{h=0}^{H} \gamma^k \nabla_\theta \log \pi_\theta(a_k|s_k) \left(Q^{\pi_\theta}(s,a,h) - b(s,h)\right)\right\},$$

known as the policy gradient theorem where the baseline $b(s,h)$ is an arbitrary function that may be used to reduce the variance, and $Q^{\pi_\theta}(s,a,h)$ represents the action-▶ value function.

While likelihood ratio gradients have been known since the late 1980s, they have recently experienced an upsurge of interest due to their demonstrated effectiveness in applications; see, e.g., Peters and Schaal (2008)), progress toward variance reduction using optimal baselines (Lawrence et al. 2003), rigorous understanding of the relationships between value functions and policy gradients (Sutton et al. 2000), policy gradients in reproducing kernel Hilbert space (Bagnell 2004), as well as faster, more robust convergence using natural policy gradients (Bagnell 2004; Peters and Schaal 2008)

A recent major development (Silver et al. 2014) demonstrates that many of the key results from model-free stochastic policy search can be transferred to deterministic policy classes by considering the limiting case of a likelihood ratio method. Importantly, this estimation of a deterministic policy gradient can be much more sample efficient than existing techniques; the caveat remains that the total return may indeed fail to be differentiable and both practical performance and theory in such settings are poorly understood.

Cross-References

- ▶ Policy Search
- ▶ Reinforcement Learning
- ▶ Value Function Approximation

Recommended Reading

Bagnell JA (2004) Learning decisions: robustness, uncertainty, and approximation. Doctoral dissertation, Robotics institute, Carnegie Mellon University, Pittsburgh

Fu MC (2006) Stochastic gradient estimation. In: Henderson SG, Nelson BL (eds) Handbook on operations research and management science: simulation, vol 19. Elsevier, Burlington, pp 575–616

Glynn P (1990) Likelihood ratio gradient estimation for stochastic systems. Commun ACM 33(10):75–84

Lawrence G, Cowan N, Russell S (2003) Efficient gradient estimation for motor control learning. In: Proceedings of the international conference on uncertainty in artificial intelligence (UAI), Acapulco

Peters J, Schaal S (2008) Reinforcement learning of motor skills with policy gradients. Neural Netw 21(4):682–697

Pontryagin LS, Boltyanskii VG, Gamkrelidze RV, Mishchenko E (1962) The mathematical theory of optimal processes. International series of monographs in pure and applied mathematics. Interscience publishers, New York

Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M (2014) Deterministic policy gradient algorithms. In: Proceedings of the 31st international conference on Machine learning (ICML), Bejing

Spall JC (2003) Introduction to stochastic search and optimization: estimation, simulation, and control. Wiley, Hoboken

Sutton RS, McAllester D, Singh S, Mansour Y (2000) Policy gradient methods for reinforcement learning with function approximation. In: Solla SA, Leen TK, Mueller KR (eds) Advances in neural information processing systems (NIPS). MIT, Denver

Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach Learn 8:229–256

Policy Search

▶ Markov Decision Processes

POMDPs

▶ Partially Observable Markov Decision Processes

POS Tagging

Walter Daelemans
CLIPS University of Antwerp, Antwerpen, Belgium

Synonyms

Grammatical tagging; Morphosyntactic disambiguation; Part of speech tagging; Tagging

Definition

Part-of-speech tagging (POS tagging) is a process in which each word in a text is assigned its appropriate morphosyntactic category (for example *noun-singular*, *verb-past*, *adjective*, *pronoun-personal*, and the like). It therefore provides information about both morphology (structure of words) and syntax (structure of sentences). This disambiguation process is determined both by constraints from the lexicon (what are the possible categories for a word?) and by constraints from the context in which the word occurs (which of the possible categories is the right one in this context?). For example, a word like *table* can be a noun-singular, but also a verb-present (as in *I table this motion*). This is lexical knowledge. It is the context of the word that should be used to decide which of the possible categories is the correct one. In a sentence like *Put it on the table,* the fact that *table* is preceded by the determiner *the*, is a good indication that it is used as a noun here. Systems that automatically assign parts of speech to words in text should take into account both lexical and contextual constraints, and they are typically found in implementations as a lookup module and a disambiguation module.

Motivation and Background

In most natural language processing (NLP) applications, POS tagging is one of the first steps to allow abstracting away from individual words. It is not to be confused with *lemmatization*, a process that reduces morphological variants of words to a canonical form (the citation form, for example, infinitive for verbs and singular for nouns). Whereas lemmatization allows abstraction over different forms of the same word, POS tagging abstracts over sets of different words that have the same function in a sentence. It should also not be confused with *tokenization*, a process that detects word forms in text, stripping off punctuation, handling abbreviations, and so on. For example, the string *don't* could be converted to *do not*. Normally, a POS tagging system would take tokenized text as input. More advanced tokenizers may even handle multiword items, for example treating *in order to* not as three separate words but as a single lexical item.

Applications. A POS tagger is the first disambiguation module in text analysis systems. In order to determine the syntactic structure of a sentence (and its semantics), we have to know the parts of speech of each word. In earlier approaches to syntactic analysis (parsing), POS tagging was part of the parsing process. However, individually trained and optimized POS taggers have increasingly become a separate module in shallow or deep syntactic analysis systems. By extension, POS tagging is also a foundational module in text mining applications ranging from information extraction and terminology/ontology extraction to summarization and question answering.

Apart from being one of the first modules in any text analysis system, POS tagging is also useful in linguistic studies (corpus linguistics) – for example for computing frequencies of disambiguated words and of superficial syntactic structures. In speech technology, knowing the part of speech of a word can help in speech synthesis (the verb "subJECT" is pronounced differently from the noun "SUBject"), and in speech recognition, POS taggers are used in some approaches to language modeling. In spelling and grammar checking, POS tagging plays a role in increasing the precision of such systems.

Part-of-speech tag sets. The inventory of POS tags can vary from tens to hundreds depending on the richness of morphology and syntax that is represented and on the inherent morphological complexity of a language. For English, the tag sets most used are those of the Penn Treebank (45 tags; Marcus et al. 1993), and the CLAWS C7 tag set (146 tags; Garside and Smith 1997). Tag sets are most often developed in the context of the construction of annotated corpora. There have been efforts to standardize the construction of tag sets to increase translatability between different tag sets, such as Eagles. (http://www.ilc.cnr.it/EAGLES96/browse.html) and ISO/TC 37/SC 4. (http://www.tc37sc4.org/)

The following example shows both tag sets. By convention, a tagged word is represented by attaching the POS tag to it, separated by a slash.

Pierre/NNP Vinken/NNP ,/, 61/CD years/NNS old/JJ ,/, will/MD join/VB the/DT board/NN as/IN a/DT nonexecutive/JJ director/NN Nov./NNP 29/CD ./. [Penn Treebank]

Pierre/NP1 Vinken/NP1 ,/, 61/MC years/NNT2 old/JJ ,/, will/VM join/VVI the/AT Board/NN1 as/II a/AT1 nonexecutive/JJ director/NN1 Nov./NPM1 29/MC ./. [CLAWS C7]

As can be seen, the tag sets differ in level of detail. For example, NNT2 in the C7 tag set indicates a plural temporal noun (as a specialization of the word class noun), whereas the Penn

Treebank tag set only specializes to plural noun (NNS).

Like most tasks in NLP, POS tagging is a disambiguation task, and both linguistic knowledge-based handcrafting methods and corpus-based learning methods have been proposed for this task. We will restrict our discussion here to the statistical and machine learning approaches to the problem, which have become mainstream because of the availability of large POS tagged corpora and because of better accuracy in general than handcrafted systems. A state of the art system using a knowledge-based approach is described in Karlsson et al. (1995).

A decade old now, but still a complete and informative book-length introduction to the field of POS tagging is van Halteren (1999). It discusses many important issues that are not covered in this article (performance evaluation, history, handcrafting approaches, tag set development issues, handling unknown words, and more.). A more recent introductory overview is Chap. 5 in Jurafsky and Martin (2008).

Statistical and Machine Learning Approaches to Tagging

In the late 1970s, statistical approaches based on n-gram probabilities (probabilities that sequences of n tags occur in a corpus) computed on frequencies in tagged corpora have already been proposed by the UCREL team at the University of Lancaster (Garside and Smith 1997). These early models lacked a precise mathematical framework and a principled solution to working with zero- or low probability frequencies. It was realized that Hidden Markov Models (HMM) in use in speech recognition were applicable to the tagging problem as well.

HMMs

HMMs are probabilistic finite state automata that are flexible enough to combine n-gram information with other relevant information to a limited extent. They allow supervised learning by computing the probabilities of n-grams from tagged corpora, and unsupervised learning using the Baum-Welch algorithm. Finding the most probable tag sequence given a sequence of words (decoding) is done using the Viterbi search. In combination with smoothing methods for low-frequency events and special solutions for handling unknown words, this approach results in a state-of-the-art tagging performance. A good implementation is TnT (Trigrams'n Tags Brants 2000).

Transformation-Based Error-Driven Learning (Brill-Tagging)

Transformation-based learning is an eager learning method in which the learner extracts a series of rules, each of which transforms a tag into another tag given a specific context. Learning starts with an initial annotation (e.g., tag each word in a text by the POS tag it is most frequently associated with in a training corpus), and compares this annotation with a gold standard annotation (annotated by humans). Discrepancies trigger the generation of rules (constrained by templates), and in each cycle, the best rule is chosen. The best rule is the one that most often leads to a correct transformation in the whole training corpus (Brill 1995a). An unsupervised learning variant (using a lexicon with word-tag probabilities) is described in Brill (1995b). Fully unsupervised POS tagging can also be achieved using distributional clustering techniques, as pioneered by Schutze (1995). However, these methods are hard to evaluate and compare to supervised approaches. The best way to evaluate them is indirectly, in an application-oriented way, as in Ushioda (1996).

Other Supervised Learning Methods

As a supervised learning task, POS tagging has been handled mostly as in a *sliding window* representation. Instances are created by making each word in each sentence a focus feature of an instance, and adding the left and right context as additional features. The class to be predicted is the POS tag of the focus word. Instead of using the words themselves as features, information about them can be used as features as well (e.g., capitalized or not, hyphenated or not, the POS tag of the word for left context words as predicted by the tagger previously, a symbol representing the possible lexical categories of the focus word and right context words, first and last letters of the word in each position, and so on.).

The following table lists the structure of instance representations for part of the sentence shown earlier. In this case the words themselves are feature values, but most often other derived features would replace these because of sparseness problems.

		Focus		Class	
=	=	Pierre	Vinken		NNP
=	Pierre	Vinken		61	NNP
Pierre	Vinken		61	years	
Vinken		61	years	old	CD

Most classification-based, supervised machine learning methods can be, and have been applied to this problem, including decision tree learning (Schmid 1994b), memory-based learning (Daelemans et al. 1996), maximum entropy models (Ratnaparkhi 1996), neural networks (Schmid 1994a), ensemble methods (van Halteren et al. 2001), and many others. All these methods seem to converge to a 96–97 % accuracy rate on the Wall Street Journal corpus using the Penn Treebank tag set. In a systematic comparison of some of the methods listed here, van Halteren et al. (2001) found that TnT outperforms maximum entropy and memory-based learning methods, which in turn outperform Brill tagging. Nonpropositional supervised learning methods have been applied to the task as well (Cussens 1997) with grammatical structure as background knowledge with similar results. The best results reported on the WSJ corpus so far is bidirectional perceptron learning (Shen et al. 2007) with a 97.33 % accuracy.

Because of these high scores, POS tagging (at least for English) is considered by many a solved problem. However, as for most machine-learning based NLP systems, domain adaptation is still a serious problem for POS tagging. A tagger trained to high accuracy on newspaper language will fail miserably on other types of text, such as medical language.

Cross-References

▶ Classification
▶ Clustering
▶ Decision Tree
▶ Document Categorization
▶ Inductive Logic Programming
▶ Information Retrieval
▶ Lazy Learning
▶ Maxent Models
▶ Text Mining

Recommended Reading

Brants T (2000) TnT – a statistical part-of-speech tagger. In: Proceedings of the sixth applied natural language processing conference ANLP-2000, Seattle

Brill E (1995a) Transformation-based error-driven learning and natural language processing: a case study in part-of-speech tagging. Comput Linguist 21(4):543–565

Brill E (1995b) Unsupervised learning of disambiguation rules for part of speech tagging. In: Proceedings of the third workshop on very large corpora. Ohio State University, Ohio, pp 1–13

Cussens J (1997) Part-of-speech tagging using progol. In: Lavrac N, Dzeroski S (eds) Proceedings of the seventh international workshop on inductive logic programming. Lecture notes in computer science, vol 1297. Springer, London, pp 93–108

Daelemans W, Zavrel J, Berck P, Gillis S (1996) MBT: a memory-based part of speech tagger generator. In: Proceedings of the fourth workshop on very large corpora, Copenhagen, pp 14–27

Garside R, Smith N (1997) A hybrid grammatical tagger: CLAWS4. In: Garside R, Leech G, McEnery A (eds) Corpus annotation: linguistic information from computer text corpora. Longman, London, pp 102–121

Jurafsky D, Martin J (2008) Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd edn. Prentice Hall, Upper Saddle River

Karlsson F, Voutilainen A, Heikkilä J, Anttila A (1995) Constraint grammar. A language-independent system for parsing unrestricted text. Mouton de Gruyter, Berlin/New York, p 430

Marcus M, Santorini B, Marcinkiewicz M (1993) Building a large annotated corpus of English: the Penn Treebank. Comput Linguist 19(2):313–330

Ratnaparkhi A (1996) A maximum entropy part of speech tagger. In: Proceedings of the ACL-SIGDAT conference on empirical methods in natural language processing, Philadelphia, pp 17–18

Schmid H (1994a) Part-of-speech tagging with neural networks. In: Proceedings of COLING-94, Kyoto, pp 172–176

Schmid H (1994b) Probabilistic part-of-speech tagging using decision trees. In: Proceedings of the international conference on new methods in language processing (NeMLaP), Manchester, pp 44–49

Schutze H (1995) Distributional part-of-speech tagging. In: Proceedings of EACL 7, Dublin, pp 141–148

Shen L, Satta G, Joshi A (2007) Guided learning for bidirectional sequence classification. In: Proceedings of the 45th annual meetings of the association of computational linguistics (ACL 2007), Prague, pp 760–767

Ushioda A (1996) Hierarchical clustering of words and applications to NLP tasks. In: Proceedings of the fourth workshop on very large corpora, Somerset, pp 28–41

van Halteren H (ed) (1999) Syntactic wordclass tagging. Kluwer Academic Publishers, Boston

van Halteren H, Zavrel J, Daelemans W (2001) Improving accuracy in NLP through combination of machine learning systems. Comput Linguist 27(2):199–229

Positive Definite

▶ Positive Semidefinite

Positive Predictive Value

▶ Precision

Positive Semidefinite

Synonyms

Positive definite

Definition

A symmetric $m \times m$ matrix K satisfying $\forall x \in c^m : x^* Kx \geq 0$ is called positive semidefinite. If the equality only holds for $x = \vec{0}$ the matrix is positive definite.

A function $k : X \times X \rightarrow c, X \neq \emptyset$, is positive (semi-) definite if for all $m \in n$ and all $x_1, \ldots, x_m \in X$ the $m \times m$ matrix $\vec{K}$ with elements $K_{ij} := k(x_i, x_j)$ is positive (semi-) definite.

Sometimes the term strictly positive definite is used instead of positive definite, and positive definite refers then to positive semidefiniteness.

Posterior

▶ Posterior Probability

Posterior Probability

Geoffrey I. Webb
Faculty of Information Technology, Monash University, Victoria, Australia

Synonyms

Posterior

Definition

In Bayesian inference, a *posterior probability* of a value x of a random variable X given a context a value y of a random variable Y, $\mathrm{P}(X = x | Y = y)$, is the probability of X assuming the value x in the context of $Y = y$. It contrasts with the ▶ prior probability, $\mathrm{P}(X = x)$, the probability of X assuming the value x in the absence of additional information.

For example, it may be that the prevalence of a particular form of cancer, *exoma*, in the population is 0.1 %, so the prior probability of exoma, P(exoma = true), is 0.001. However, assume 50 % of people who have skin discolorations of greater than 1 cm width (sd > 1 cm) have exoma. It follows that the posterior probability of exoma given sd >1 cm, P(exoma = true | sd >1 cm = true), is 0.500.

Cross-References

▶ Bayesian Methods

Post-pruning

Definition

Post-pruning is a ▶ Pruning mechanism that first learns a possibly ▶ Overfitting hypothesis and then tries to simplify it in a separate learning phase.

Cross-References

▶ Overfitting
▶ Pre-pruning
▶ Pruning

Postsynaptic Neuron

The neuron that receives signals via a synaptic connection. A chemical synaptic connection between two neurons allows to transmit signals from a presynaptic neuron to a postsynaptic neuron.

Precision

Kai Ming Ting
Federation University, Mount Helen, VIC, Australia

Synonyms

Positive predictive value

Precision, Table 1 The outcomes of classification into positive and negative classes

		Assigned class	
		Positive	Negative
Actual class	Positive	True positive (TP)	False negative (FN)
	Negative	False positive (FP)	True negative (TN)

Definition

Precision is defined as the ratio of true positives (TP) and the total number of positives predicted by a model. This is defined with reference to a special case of the ▶ confusion matrix, with two classes: one designated the *positive* class and the other the *negative* class, as indicated in Table 1.

Precision can then be defined in terms of true positives and false positives (FP) as follows.

Precision = TP/(TP + FP)

Cross-References

▶ Precision and Recall

Precision and Recall

Kai Ming Ting
Federation University, Mount Helen, VIC, Australia

Definition

▶ Precision and recall are the measures used in the information retrieval domain to measure how well an information retrieval system retrieves the relevant documents requested by a user. The measures are defined as follows:

Precision = Total number of documents retrieved that are relevant/total number of documents that are retrieved

Recall = Total number of documents retrieved that are relevant/total number of relevant documents in the database

Precision and Recall, Table 1 The outcomes of classification into positive and negative classes

		Assigned class	
		Positive	Negative
Actual class	Positive	True positive (TP)	False negative (FN)
	Negative	False positive (FP)	True negative (TN)

We can employ the same terminology used in a ▶ confusion matrix to define these two measures. Let relevant documents be positive examples and irrelevant documents, negative examples. The two measures can be redefined with reference to a special case of the confusion matrix, with two classes: one designated the *positive* class and the other the *negative* class, as indicated in Table 1.

Precision = True positives/total number of positives predicted = TP/(TP + FP)

Recall = True positives/total number of actual positives = TP/(TP + FN)

Instead of two measures, they are often combined to provide a single measure of retrieval performance called the ▶ F-measure as follows:

F-measure = 2 * recall * precision/(recall + precision)

Cross-References

▶ Confusion Matrix

Predicate

A *predicate* or predicate symbol is used in logic to denote properties and relationships. Formally, if P is a predicate with arity n, and $t_1, \ldots, t_n$ is a sequence of n terms (i.e., constants, variables, or compound terms built from function symbols), then $P(t_1, \ldots, t_n)$ is an atomic formula or *atom*. Such an atom represents a statement that can be either true or false. Using logical connectives, atoms can be combined to build well-formed formulae in ▶ first-order logic or ▶ clauses in ▶ logic programs.

Cross-References

▶ Clause
▶ First-Order Logic
▶ Logic Program

Predicate Calculus

▶ First-Order Logic

Predicate Invention

Definition

Predicate invention is used in ▶ inductive logic programming to refer to the automatic introduction of new relations or predicates in the hypothesis language. Inventing relevant new predicates is one of the hardest tasks in machine learning, because there are so many possible ways to introduce such predicates and because it is hard to judge their quality. As an example, consider a situation where in the predicates `fatherof` and `motherof` are known. Then it would make sense to introduce a new predicate that is true whenever `fatherof` or `motherof` is true. The new predicate that would be introduced this way corresponds to the `parentof` predicate. Predicate invention has been introduced in the context of inverse resolution.

Cross-References

▶ Inductive Logic Programming
▶ Logic of Generality

Predicate Logic

▶ First-Order Logic

Prediction with Expert Advice

▶ Online Learning

Predictive Software Models

▶ Predictive Techniques in Software Engineering

Predictive Techniques in Software Engineering

Jelber Sayyad Shirabad
University of Ottawa, Ottawa, ON, Canada

Synonyms

Predictive software models

Introduction

Software engineering (SE) is a knowledge- and decision-intensive activity. From the initial stages of the software life cycle (i.e., requirement analysis), to the later stage of testing the system, and finally maintaining the software through its operational life, decisions need to be made which impact both its success and failure. For instance, during project planning one needs to be able to forecast or predict the required resources to build the system. At the later stages such as testing or maintenance it is desirable to know which parts of the system may be impacted by a change, or are more risky or will require more intensive testing.

The process of developing software can potentially create a large amount of data and domain knowledge. The nature of the data, of course, depends on the phase in which the data were generated. During the requirement analysis, this data most times is manifested in the form of documentations. As the process moves forward, other types of artifacts such as code and test cases are generated. However, what, when, how accurately, and how much is recorded varies from one organization to the next. More mature organizations have a tendency to maintain larger amount of data about the software systems they develop.

The data generated as part of the software engineering process captures a wide range of latent knowledge about the system. Having such a source of information, the question one needs to ask is that whether there is any technology that can leverage this potentially vast amount of data to:

- Better understand a system
- Make more informative decisions as needed through the life of an existing system
- Apply lessons learned from building other systems to the creation of a new system

As this chapter will show, machine learning (ML), which provides us with a host of algorithms and techniques to learn from data, is such a technology. In preparing this entry we have drawn from over two decades of research in applying ML to various software engineering problems. The number of potential uses of ML in SE is practically enormous and the list of applications is expanding over time. The focus of this chapter is a subset of these applications, namely the ones that aim to create models for the purpose of making a prediction regarding some aspect of a software system. One could dedicate a separate article for some of these prediction tasks, as there is a large body of research covering different aspects of interest, such as algorithms, estimation methods, features used, and the like. However, due to space constraints, we will only mention a few representative research examples. The more general topic of the application of ML in SE can be studied from different points of view. A good discussion of many such aspects and applications can be found in Zhang and Tsai (2003).

Traditionally, regression-based techniques have been used in software engineering for building predictive models. However, this requires making a decision as to what kind of regression method should be used (e.g., linear or quadratic), or alternatively what kind of curve should be fit to the data. This means that the general shape of the function is determined first, and then the model is built. Some researcher, have used ML as a way to delegate such decisions to the algorithm. In other words, it is the algorithm that would produce the best fit to the data. Some of the most common replacements in the case of regression problems have been neural networks (NN) and genetic programming (GP). However, obviously the use of such methods still requires other types of decisions, such as the topology of the network, the number of generations, or the probability of mutations to be made by humans. Sometimes, a combination of different methods such as genetic algorithms and neural networks are used, where one method explores possible parameters for the actual method used to build the model.

Software engineering-related datasets, similar to many other real world datasets, are known to contain noise. Another justification for the use of ML in software engineering applications is that it provides algorithms that are less sensitive to noise.

The Process of Applying ML to SE

To apply ML to SE, similar to other applications, one needs to follow certain steps, which include:

Understanding the problem. This is an essential step that heavily influences the decisions to follow. Examples of typical problems in the software engineering domain are the need to be able to estimate the cost or effort involved in developing a software, or to be able to characterize the quality of a software system, or to be able to predict what modules in a system are more likely to have a defect.

Casting the original problem as a learning problem. To use ML technology, one needs to decide on how to formulate the problem as a learning task. For instance, the problem of finding modules that are likely to be faulty can be cast as a classification problem, (e.g., is the module faulty or not) or a numeric prediction problem (e.g., what the estimated fault density of a module is). This mapping is not always straightforward, and may require further refinement of the original problem statement or breaking down the original problem into sub-problems, for some of them ML may provide an appropriate solution.

Collection of data and relevant background knowledge. Once the ML problem for a particular SE application is identified, one needs to collect the necessary data and background knowledge in support of the learning task. In many SE applications data is much more abundant or easier to collect than the domain theory or background knowledge relevant to a particular application. For instance, collecting data regarding faults discovered in a software system and changes applied to the source to correct a fault is a common practice in software projects. On the other hand, there is no comprehensive and agreed upon domain theory describing software systems. Having said that, in the case of some applications, if we limit ourselves to incomplete background knowledge, then it can be automatically generated by choosing a subset that is considered to be relevant. For instance, in Cohen and Devanbu (1999), the authors apply inductive logic programming to the task of predicting faulty modules in a software system. They describe the software system in terms of cohesion and coupling-based relations between classes, which are generated by parsing the source code.

Data preprocessing and encoding. Preprocessing the data includes activities such as reducing the noise, selecting appropriate subsets of the collected data, and determining a proper subset of features that describe the concept to be learned. This cleaner data will be input to a specific algorithm and implementation. Therefore, the data and background knowledge, if any, may need to be described and formatted in a manner that complies with the requirements of the algorithm used.

Applying machine learning and evaluating the results. Running a specific ML algorithm is fairly straightforward. However, one needs to measure the goodness of what is learned. For instance, in the case of classification problems, models are frequently assessed in terms of their accuracy by using methods such as holdout and cross-validation. In case of numeric prediction, other standard measures such as mean magnitude of relative error (MMRE) are commonly used. Additionally, software engineering researchers have sometimes adopted other measures for certain applications. For instance PRED(x), which is percentage of the examples (or samples) with magnitude of relative error (MRE)$\leq x$. According to Pfleeger and Atlee (2003), most managers use PRED(25) to assess cost, effort, and schedule models, and consider the model to function well if the value of PRED(25) is greater than 75 %. As for MMRE, a value of less than 25 % is considered to be good; however, other researchers, such as Boehm, would recommend a value of 10 % or less. Assessing the usefulness of what is learned sometimes requires feedback from domain experts or from end users. If what is learned is determined to be inadequate, one may need to either retry this step by adjusting the parameters of the algorithms used, or reconsider the decisions made in earlier stages and proceed accordingly.

Field testing and deployment. Once what is learned is assessed to be of value, it needs to actually be used by the intended users (e.g., project managers and software engineers). Unfortunately, despite the very large body of research in software engineering in general and use of ML in specific applications in SE, the number of articles discussing the actual use and impact of the research in industry is relatively very small. Very often, the reason for this is the lack of desire to share what the industry considers to be confidential information. However, there are numerous research articles that are based on industrial data, which is an indication of the practical benefits of ML in real-world SE.

Applications of Predictive Models in SE

The development of predictive models is probably the most common application of ML in software engineering. This observation is consistent with findings of previous research (Zhang and Tsai 2003). In this section, we mention some of the predictive models one can learn from software engineering data. Our goal is to provide examples of both well established and newer applications. It should be noted that the terminology used by researchers in the field is not always consistent. As such, one may argue that some of these examples belong to more than one category. For instance, in Fenton and Neil (1999) the authors consider predicting faults as a way of estimating software quality and maintenance effort. The paper could potentially belong to any of the categories of fault, quality, or maintenance effort prediction.

Software Size Prediction

Software size estimation is the process of predicting the size of a software system. As software size is usually an input to models that estimate project cost schedule and planning, an accurate estimation of software size is essential to proper estimation of these dependent factors. Software size can be measured in different ways, most common of which is the number of lines of code (LOC); however, other alternatives, such as function points, which are primarily for effort estimation, also provide means to convert the measure to LOC. There are different methods for software sizing, one of which is the component-based method (CBM). In a study to validate the CBM method, Dolado (2000) compared models generated by multiple ▸ linear regression (MLR) with the ones obtained by neural networks and genetic programming. He concluded that both NN- and GP-based models perform as well or better than the MLR models. One of the cited benefits of NN was its ability to capture nonlinear relations, which is one of the weaknesses of MLR, while GP was able to generate models that were interpretable. Regolin et al. (2003) also used NN- and GP-based models to predict software size in terms of LOC. They use both function points and number of components metrics for this task. Pendharkar (2004) uses decision tree regression to predict the size of OO components. The total size of the system can be calculated after the size of its components is determined.

Software Quality Prediction

The ISO 9126 quality standard decomposes quality to functionality, reliability, efficiency, usability, maintainability, and portability factors. Other models such as McCall's, also define quality in terms of factors that are themselves composed of quality criteria. These quality criteria are further associated with measurable attributes called quality metrics, for instance fault or change counts (Fenton and Pfleeger 1998) However, as stated in Fenton and Pfleeger (1998), many software engineers have a narrower view of quality as the lack of software defects. A de facto standard for software quality is fault density. Consequently, it is not surprising to see that in many published articles the problem of predicting the quality of a system is formulated as prediction of faults. To that end, there has been a large body of work over the years that has applied various ML techniques to build models to assess the quality of a system. For instance, Evett and Khoshgoftar (1998) used genetic programming to build models that predict the number of faults expected in each module. Neural networks have appeared in a number of software quality modeling applications such as Khoshgoftaar et al. (1997), which applied the technique to a large industrial system to classify modules as fault-prone or not fault-prone, or Quah and Thwin (2003) who used object-oriented design metrics as features in developing the model. In El Emam et al. (2001) the authors developed fault prediction models for the purpose of identifying high-risk modules. In this study, the authors investigated the effect of various parameter settings on the accuracy of these models. The models were developed using data from a large real-time system. More recently, Xing et al. (2005) used SVMs and Seliya and Khoshgoftaar (2007) used an EM semi-supervised learning algorithm to develop software quality models. Both these works cite the ability of these algorithms to generate models with good performance in the presence of a small amount of labeled data.

Software Cost Prediction

Software cost prediction typically refers to the process of estimating the amount of effort needed to develop a software system. As this definition suggests, cost and effort estimations are often used interchangeably. Various kinds of cost estimations are needed throughout the software life cycle. Early estimation allows one to determine the feasibility of a project. More detailed estimation allows managers to better plan for the project. As there is less information available in the early stages of the project, early predictions have a tendency to be the least accurate. Software cost and effort estimation models are among some of the oldest software process prediction models. There are different methods of estimating costs including:

(1) Expert opinion; (2) analogy based on similarity to other projects; (3) decomposition of the project in terms of components to deliver or tasks to accomplish, and to generate a total estimate from the estimates of the cost of individual components or activities; and (4) the use of estimation models (Fenton and Pfleeger 1998).

In general, organization-specific cost estimation datasets tend to be small, as many organizations deal with a limited number of projects and do not systematically collect process level data, including the actual time and effort expenditure for completion of a project. As cost estimation models are numeric predictors, many of the original modeling techniques were based on regression methods.

The study in Briand et al. (1999) aims to identify methods that generate more accurate cost models, as well as to investigate the effects of the use of organization-specific versus multi-organization datasets. The authors compared the accuracy of models generated by using ordinary least squares regression, stepwise ANOVA, CART, and analogy. The measures used were MMRE, median of MRE (MdMRE), and PRED(25). While their results did not show a statistical difference between models obtained from these methods, they suggest that CART models are of particular interest due to their simplicity of use and interpretation.

Shepperd and Schofield (1997) describes the use of analogies for effort prediction. In this method, projects are characterized in terms of attributes such as the number of interfaces, the development method, or the size of the functional requirements document. The prediction for a specific project is made based on the characteristics of projects most similar to it. The similarity measure used in Shepperd and Schofield (1997) is Euclidean distance in n-dimensional space of project features. The proposed method was validated on nine different industrial datasets, covering a total of 275 projects. In all cases, the analogy-based method outperforms algorithmic models based upon stepwise regression when measured in terms of MMRE. When results are compared using PRED(25) the analogy-based method generates more accurate models in seven out of nine datasets. Decision tree and neural network-based models are also used in a number of studies on effort estimation models.

In a more recent paper, (Oliveira 2006), a comparative study of support vector regression (SVR), radial basis function ▶ neural networks (RBFNs), and ▶ linear regression-based models for estimation of a software project effort is presented. Both linear as well as RBF kernels were used in the construction of SVR models. Experiments using a dataset of software projects from NASA showed that SVR significantly outperforms RBFNs and linear regression in this task.

Software Defect Prediction

In research literature one comes across different definitions for what constitutes a defect: fault and failure. According to Fenton and Pfleeger (1998) a fault is a mistake in some software product due to a human error. Failure, on the other hand, is the departure of the system from its required behavior. Very often, defects refer to faults and failures collectively. In their study of defect prediction models, Fenton and Neil observed that, depending on the study, defect count could refer to a post-release defect, the total number of known defects, or defects that are discovered after some arbitrary point in the life cycle. Additionally, they note that defect rate, defect density, and failure rate are used almost interchangeably in the literature (Fenton and Neil 1999). The lack of an agreed-upon definition for such a fundamental measure makes comparison of the models or published results in the literature difficult. Two major reasons cited in research literature for developing defect detection models are assessing software quality and focusing testing or other needed resources on modules that are more likely to be defective. As a result, we frequently find ourselves in a situation where a model could be considered both a quality prediction model and a defect prediction model. Therefore, most of the publications we have mentioned under software quality prediction could also be referred to in this subsection. Fenton and Neil suggest using Bayesian Belief Networks as an alternative to other existing methods (Fenton and Neil 1999).

Software Reliability Prediction

The ANSI Software Reliability Standard defines software reliability as:

> “the probability of failure-free operation of a computer program for a specified time in a specified environment.”

Software reliability is an important attribute of software quality. There are a number of publications on the use of various neural network-based reliability prediction models, including Sitte (1999) where NN-based software reliability growth models are compared with models obtained through recalibration of parametric models. Results show that neural networks are not only much simpler to use than the recalibration method, but that they are equal or better trend predictors. In Pai and Hong (2006) the authors use SVMs to predict software reliability. They use simulated annealing to select the parameters of the SVM model. Results show that an SVM-based model with simulated annealing performs better than existing Bayesian models.

Software Reusability Prediction

The use of existing software artifacts or software knowledge is known as software reuse. The aim of software reuse is to increase the productivity of software developers, and increase the quality of end product, both of which contribute to overall reduction in software development costs. While the importance of software reuse was recognized as early as 1968 by Douglas McIlroy, applications of ML in predicting reusable components are relatively few and far between. The typical approach is to label the reusable piece of code (i.e., a module or a class) as one of reusable or non-reusable, and to then use software metrics to describe the example of interest. An early work by Esteva (1990) used ID3 to classify Pascal modules from different application domains as either reusable or not-reusable. These modules contained different number of procedures. Later work in Mao et al. (1998) uses models built using C4.5 as a means to verify three hypothesis of correlation between reusability and the quantitative attributes of a piece of software: inheritance, coupling, and complexity. For each hypothesis, a set of relevant metrics (e.g., complexity metrics for a hypothesis on the relation between complexity and reuse) is used to describe examples. Each example is labeled as one of four classes of reusability, ranging from “totally reusable” to “not reusable at all.” If the learned model performs well then this is interpreted as the existence of a hypothesized relation between reuse and one of the abovementioned quantitative attributes.

Other Applications

In this section, we discuss some of the more recent uses of ML techniques in building predictive models for software engineering applications that do not fall into one the above widely researched areas.

In Padberg et al. (2004) models are learned to predict the defect content of documents after software inspection. Being able to estimate how many defects are in a software document (e.g., specifications, designs) after the inspection, allows managers to decide whether to re-inspect the document to find more defects or pass it on to the next development step. To capture the non-linear relation between the inspection process metrics, such as total number of defects found by the inspection team and the number of defects in the document, the authors train a neural network. They conclude that these models yield much more accurate estimates than standard estimation methods such as capture-recapture and detection profile.

Predicting the stability of object-oriented software, defined as the ease by which a software system or component can be changed while maintaining its design, is the subject of research in Grosser et al. (2002). More specifically, stability is defined as preservation of the class interfaces through evolution of the software. To accomplish the above task, the authors use Cased-Base Reasoning. A class is considered stable if its public interface in revision J is included in revision $J + 1$. Each program class or case is represented by structural software metrics, which belong to one of the four categories of coupling, cohesion, inheritance, and complexity.

Models that predict which defects will be escalated are developed in Ling et al. (2006). Escalated defects are the ones that were not addressed prior to release of the software due to factors such as deadlines and limited resources. However, after the release, these defects are escalated by the customer and must be immediately resolved by the vendor at a very high cost. Therefore, the ability to predict the risk of escalation for existing defect reports will prevent many escalations, and result in large savings for the vendor. The authors in this paper show how the problem of maximizing net profit (the difference in cost of following predictions made by the escalation prediction model versus the existing alternative policy) can be converted to cost-sensitive learning. The assumption here is that net profit can be represented as a linear combination of true positive, false positive, true negative, and false negative prediction counts, as is done for cost-sensitive learning that attempts to minimize the weighted cost of the abovementioned four factors. The results of the experiments performed by the authors show that an improved version of the CSTree algorithm can produce comprehensible models that generate a large positive unit net profit.

Most predictive models developed for software engineering applications, including the ones cited in this article, make prediction regarding a single entity – for instance, whether a module is defective, how much effort is needed to develop a system, is a piece of code reusable, and so on. Sayyad Shirabad et al. (2007) introduced the notion of relevance relations among multiple entities in software systems. As an example of such relations, the authors applied historic problem report and software change data to learned models for the Co-update relation among files in a large industrial telecom system. These models predict whether changing one source file may require a change in another file. Different sets of attributes, including syntax-based software metrics as well as textual attributes such as source file comments and problem reports, are used to describe examples of the Co-update relation. The C5.0 decision tree induction algorithm was used to learn these predictive models. The authors concluded that text-based attributes outperform syntactic attributes in this model-building task. The best results are obtained for text-based attributes extracted from problem reports. Additionally, when these attributes are combined with syntactic attributes, the resulting models perform slightly better.

Future Directions

As we mentioned earlier due to its decision-intensive nature, there is potential for learning a large number of predictive models for software engineering tasks. A very rich area of research for future applications of predictive models in software engineering is in *Autonomic Computing*. Autonomic computing systems, as was put forward in Ganek and Corbi (2003), should be:

- *Self-configuring*: able to adapt to changes in the system in a dynamic fashion.
- *Self-optimizing*: able to improve performance and maximize resource allocation and utilization to meet end users' needs while minimizing human intervention.
- *Self-healing*: able to recover from mistakes by detecting improper operations proactively or reactively and then initiate actions to remedy the problem without disrupting system applications.
- *Self-protecting*: able to anticipate and take actions against intrusive behaviors as they occur, so as to make the systems less vulnerable to unauthorized access.

Execution of actions in support of the capabilities mentioned above follows the detection of a triggering change of state in the environment. In some scenarios, this may entail a prediction about the current state of the system; in other scenarios, the prediction may be about the future state of the system. In a two-state scenario, the system needs to know whether it is in a normal or abnormal (undesired) state. Examples of undesired states are *needs optimization* or *needs healing*. The detection of the state of a system can be cast as a classification problem. The decision as to what attributes should be used to represent each example of a normal or an abnormal state depends on the specific prediction model that we would like to build and on the monitoring capabilities of the system. Selecting the best attributes among a set of potential attributes will require empirical analysis. However, the process can be further aided by:

- *Expert knowledge*: Based on their past experience, hardware and software experts typically have evidence or reasons to believe that some attributes are better indicators of desired or undesired states of the system.
- *Documentation*: System specification and other documents sometimes include the range of acceptable values for certain parameters of the system. These parameters could be used as attributes.
- *Feature selection*: This aims to find a subset of available features or attributes that result in improving a predefined measure of goodness, such as the accuracy of the model. Reducing the number of features may also result in a simpler model. One of the benefits of such simpler models is the higher prediction speed, which is essential for timely responses by the autonomic system to changes in the environment.

Obviously, given enough examples of different system states, one can build multi-class models, which can make finer predictions regarding the state of the system.

In the context of autonomic computing, besides classification models, numeric predictors can also be used for resource estimation (e.g., what is the appropriate database cache size considering the current state of the system). Furthermore, an autonomic system can leverage the ability to predict the future value of a variable of interest, such as the use of a particular resource based on its past values. This can be accomplished through ▶ time series predictions. Although researchers have used neural networks and support vector machines for time series prediction in various domains, we are not aware of an example of the usage of such algorithms in autonomic computing.

Recommended Reading

Briand L, El Emam K, Surmann D, Wieczorek I (1999) An assessment and comparison of common software cost estimation modeling techniques. In: Proceedings of 21st international conference on software engineering, Los Angeles, pp 313–322

Cohen W, Devanbu P (1999) Automatically exploring hypotheses about fault prediction: a comparative study of inductive logic programming methods. Int J Softw Eng Knowl Eng 9(5):519–546

Dolado JJ (2000) A validation of the component-based method for software size estimation. IEEE Trans Softw Eng 26(10):1006–1021

El Emam K, Benlarbi S, Goel N, Rai S (2001) Comparing case-based reasoning classifiers for predicting high risk software components. J Syst Softw 55(3): 301–320

Esteva JC (1990) Learning to recognize reusable software modules using an inductive classification system. In: Proceedings of the fifth Jerusalem conference on information technology, Jerusalem, pp 278–285

Evett M, Khoshgoftar T (1998) GP-based software quality prediction. In: Proceedings of the third annual conference on genetic programming, pp 60–65

Fenton N, Neil M (1999) A critique of software defect prediction models. IEEE Trans Softw Eng 25(5):675–689

Fenton NE, Pfleeger SL (1998) Software metrics: a rigorous and practical approach, 2nd edn. PWS, Boston

Ganek AG, Corbi TA (2003) The dawning of autonomic computing era. IBM Syst J 42(1):5–18

Grosser D, Sahraoui HA, Valtchev P (2002) Predicting software stability using case-based reasoning. In: Proceedings of 17th IEEE international conference on automated software engineering (ASE), Edinburgh, pp 295–298

Khoshgoftaar T, Allen E, Hudepohl J, Aud S (1997) Applications of neural networks to software quality modeling of a very large telecommunications system. IEEE Trans Neural Netw 8(4):902–909

Ling C, Sheng V, Bruckhaus T, Madhavji N (2006) Maximum profit mining and its application in software development. In: Proceedings of the 12th ACM international conference on knowledge discovery and data mining (SIGKDD), Philadelphia, pp 929–934

Mao Y, Sahraoui H, Lounis H (1998) Reusability hypothesis verification using machine learning techniques: a case study. In: Proceedings of the 13th IEEE international conference on automated software engineering, Honolulu, pp 84–93

Oliveira A (2006) Estimation of software project effort with support vector regression. Neurocomputing 69(13–15):1749–1753

Padberg F, Ragg T, Schoknecht R (2004) Using machine learning for estimating the defect content after an inspection. IEEE Trans Softw Eng 30(1):17–28

Pai PF, Hong WC (2006) Software reliability forecasting by support vector machines with simulated annealing algorithms. J Syst Softw 79(6):747–755

Pendharkar PC (2004) An exploratory study of object-oriented software component size determinants and the application of regression tree forecasting models. Inf Manag 42(1):61–73

Pfleeger SL, Atlee JM (2003) Software engineering: theory and practice. Prentice-Hall, Upper Saddle River

Quah TS, Thwin MMT (2003) Application of neural networks for software quality prediction using object-oriented metrics. In: Proceedings of international conference on software maintenance, Amsterdam, pp 22–26

Regolin EN, de Souza GA, Pozo ART, Vergilio SR (2003) Exploring machine learning techniques for software size estimation. In: Proceedings of the 23rd international conference of the Chilean computer science society (SCCC), Chillan, pp 130–136

Sayyad Shirabad J, Lethbridge TC, Matwin S (2007) Modeling relevance relations using machine learning techniques. In: Tsai J, Zhang D (eds) Advances in machine learning applications in software engineering. IGI, pp 168–207

Seliya N, Khoshgoftaar TM (2007) Software quality estimation with limited fault data: a semi-supervised learning perspective. Softw Qual J 15(3):327–344

Shepperd M, Schofield C (1997) Estimating software project effort using analogies. IEEE Trans Softw Eng 23(11):736–743

Sitte R (1999) Comparison of software-reliability-growth predictions: neural networks vs parametric-recalibration. IEEE Trans Reliab 48(3):285–291

Xing F, Guo P, Lyu MR (2005) A novel method for early software quality prediction based on support vector machine. In: Proceedings of IEEE international conference on software reliability engineering, Chicago, pp 213–222

Zhang Du, Tsai JP (2003) Machine learning and software engineering. Softw Qual J 11(2):87–119

Preference Learning

Johannes Fürnkranz[1,3] and Eyke Hüllermeier[2]
[1]Knowledge Engineering Group, TU Darmstadt, Darmstadt, Deutschland
[2]Department of Computer Science, Paderborn University, Paderborn, Germany
[3]Department of Information Technology, University of Leoben, Leoben, Austria

Abstract

Preference learning refers to the task of learning to predict (contextualized) preferences on a collection of alternatives, which are often represented in the form of an order relation, on the basis of observed or revealed preference information. Supervision in preference learning is typically weak, in the sense that only partial information about sought structures or indirect information about an underlying value function are provided; a common example is feedback in the form of pairwise comparisons between alternatives. Especially important in preference learning are ranking problems, in which preferences are represented in terms of total or partial order relations. Such problems can be approached in two fundamentally different ways, either by learning binary preferences on pairs of alternatives or by inducing an underlying (latent) value function on single alternatives.

Synonyms

Comparison training; Constraint classification; Learning from preferences

Motivation and Background

Preference information plays a key role in automated decision-making and appears in various guises in artificial intelligence (AI) research (Domshlak et al. 2011). In particular, the formal modeling of preferences can be considered an essential aspect of autonomous agent design. Yet, in spite of the existence of formalisms for representing preferences in a compact way, such as CP-networks (Boutilier et al. 2004), modeling preferences by hand is a difficult task. This is an important motivation for preference *learning*, which is meant to support and partly automatize the design of preference models. Roughly speaking, preference learning is concerned with the automated acquisition of preference models from observed or revealed preference information, that is, data from which (possibly uncertain) preference representations can be deduced in a direct or indirect way.

Computerized methods for revealing the preferences of individuals (users) are useful not only in AI but also in many other fields showing a tendency for *personalization* of products and services, such as computational advertising, e-commerce, and information retrieval, where such techniques are also known as ▸ learning to rank (Liu 2011). Correspondingly, a number of methods and tools have been proposed with the goal of leveraging the manifold information that users provide about their preferences, either explicitly via ratings, written reviews, etc. or implicitly via their behavior (shopping decisions, websites visited, and so on). Typical examples include ▸ recommender systems and ▸ collaborative filtering, which can be viewed as special cases of preference learning. A first attempt at setting a common framework for this emerging subfield of machine learning was made by Fürnkranz and Hüllermeier (2010).

Ranking is one of the key tasks in the realm of preference learning. One can distinguish between two important types of ranking problems, namely, learning from object and learning from label preferences. A ranking is a special type of preference structure, namely, a *strict total order*, that is, a binary relation $\succ$ on a set $\mathcal{A}$ of alternatives that is total, irreflexive, and transitive. In agreement with our preference semantics, $a \succ b$ suggests that alternative a is preferred to alternative b. However, in a wider sense, the term "preference" can simply be interpreted as any kind of order relation. For example, $a \succ b$ can also mean that a is an algorithm that outperforms b on a certain problem or that a is a student finishing her studies before another student b.

Structure of the Learning System

An important difference between object and label ranking concerns the formal representation of the preference context and the alternatives to be ordered. In object ranking, the alternatives themselves are characterized by properties, typically in terms of a feature vector (attribute-value representation). Thus, the learner has the possibility to generalize via properties of the alternatives, whence a ranking model can be applied to arbitrary sets of such alternatives. In label ranking, the alternatives to be ranked are labels as in classification learning, i.e., mere identifiers without associated properties. Instead, the ranking context is characterized in terms of an instance from a given instance space, and the task of the model is to rank alternatives depending on properties of the context. Thus, the context may change (as opposed to object ranking, where it is implicitly fixed), but the objects to be ranked remain the same. Stated differently, object ranking is the problem to rank varying sets of objects under invariant preferences, whereas label ranking is the problem to rank an invariant set of objects under varying preferences.

Both problems can be approached in two principal ways, either by learning a *value function* that induces the sought ranking by *evaluating individual alternatives* or by comparing pairs of alternatives, that is, learning a *binary preference relation*. Note that the first approach implicitly assumes an underlying total order relation, since numerical (or at least totally ordered) utility scores enforce the comparability of alternatives. The second approach is more general in this regard, as it also allows for partial order relations. On the other hand, this approach may lead to additional complications, since a set of *hypothetical* binary preferences induced from empirical data may exhibit inconsistencies in the form of preferential cycles.

Learning from Object Preferences

The most frequently studied problem in learning from preferences is to induce a *ranking function* $r(\cdot)$ that is able to order any (finite) subset $\mathcal{O}$ of an underlying (possibly infinite) class $\mathcal{X}$ of objects. That is, $r(\cdot)$ assumes as input a subset $\mathcal{O} \subseteq \mathcal{X}$ of objects and returns as output a permutation τ of $\{1, \ldots, |\mathcal{O}|\}$. The interpretation of this permutation is that, for objects $x_i, x_j \in \mathcal{O}$, the former is preferred to the latter whenever $\tau(i) < \tau(j)$. The objects themselves are typically characterized by a finite set of features as in conventional attribute-value learning. The training data consists of a set of exemplary pairwise preferences $x \succ x'$ with $x, x' \in \mathcal{X}$. A survey of object ranking approaches is given by Kamishima et al. (2010).

Note that, in order to evaluate the predictive performance of a ranking algorithm, an accuracy measure (or loss function) is needed that compares a predicted ranking with a given reference ranking. To this end, one can refer, for example, to statistical measures of ▸ rank correlation. Expected or empirical loss minimization is a difficult problem for measures of that kind, especially because they are not (instance-wise) decomposable.

Many ▸ learning to rank problems may be viewed as object ranking problems. For example, Joachims (2002) studies a scenario where the training information could be provided implicitly by the user who clicks on some of the links in a query result and not on others. This information can be turned into binary preferences by assuming a preference of the selected pages over those nearby pages that are not clicked on. Applications in information retrieval typically suggest loss functions that put more emphasis on the top and less on the bottom of a ranking; for this purpose, specific measures have been proposed, such as the (normalized) discounted cumulative gain (Liu 2011).

Learning from Label Preferences

In label ranking, preferences are contextualized by elements x of an instance space $\mathcal{X}$, and the goal is to learn a ranking function $\mathcal{X} \longrightarrow \mathcal{S}_m$ for a fixed $m \geq 2$. Thus, for any instance $x \in \mathcal{X}$ (e.g., a person), a prediction in the form of an associated ranking $\succ_x$ of a finite set $\mathcal{L} = \{\lambda_1, \ldots, \lambda_m\}$ of labels or alternatives is sought, where $\lambda_i \succ_x \lambda_j$ means that instance x prefers λ_i to λ_j. Again, the quality of a prediction of that kind is typically captured in terms of a rank correlation measure (or an associated loss function). The training information consists of a set of instances for which (partial) knowledge about the associated preference relation is available. More precisely, each training instance x is associated with a subset of all pairwise preferences. Thus, despite the assumption of an underlying ("true") target ranking, the training data is not expected to provide full information about such rankings (and may even contain inconsistencies, such as pairwise preferences that are conflicting due to observation errors).

The above formulation essentially follows Fürnkranz and Hüllermeier (2010), though similar formalizations have been proposed independently by several authors; for an overview, see the survey papers by Vembu and Gärtner (2010) and Zhou et al. (2014). Label ranking contributes to the general trend of extending machine learning methods to complex and structured output spaces (Tsochantaridis et al. 2005). Moreover, label ranking can be viewed as a generalization of several standard learning problems. In particular, the following well-known problems are special cases of learning label preferences: (i) ▸ classification, where a single class label λ is assigned to each instance x; this is equivalent to the set of preferences $\{\lambda \succ_x \lambda_j \,|\, \lambda_j \in \mathcal{L} \setminus \{\lambda\}\}$, and (ii) ▸ multi-label classification, where each training example x is associated with a subset $L \subseteq \mathcal{L}$ of possible labels. This is equivalent to the set of preferences $\{\lambda_i \succ_x \lambda_j \,|\, \lambda_i \in L, \lambda_j \in \mathcal{L} \setminus L\}$. In each of the former scenarios, the sought prediction can be obtained by post-processing the output of a ranking model $f : \mathcal{X} \longrightarrow \mathcal{S}_m$ in a suitable way. For example, in multi-class classification, where only a single label is requested, it suffices to project a label ranking to the top-ranked label.

Applications of this general framework can be found in various fields, for example, in marketing research; here, one might be interested in discovering dependencies between properties of clients and their preferences for products. Another application scenario is ▸ meta-learning, where the task is to rank learning algorithms according to their suitability for a new dataset, based on the characteristics of this dataset (Schäfer and Hüllermeier 2015). Moreover, every preference statement in the well-known CP-nets approach (Boutilier et al. 2004), a qualitative graphical representation that reflects conditional dependence and independence of preferences under a ceteris paribus interpretation, formally corresponds to a label ranking function that orders the values of a certain attribute depending on the values of the parents of this attribute (predecessors in the graph representation).

Other Settings

A number of variants of the above ranking problems have been proposed and studied in the literature. For example, a setting referred to as *instance ranking* is very similar to object ranking. However, instead of relative (pairwise) comparisons, training data consists of absolute ratings of alternatives; typically these ratings are taken from an ordinal scale, such as 1 to 5 stars. Moreover, a predicted ranking is not compared with another (ground-truth) ranking but with the multipartition induced by the rating of the alternatives (Fürnkranz et al. 2009).

Attempts have also been made at combining object and label ranking, that is, to exploit feature representations of both the preference context and the alternatives to be ranked. One approach is to combine both pieces of information by means of a *joint feature map* $\phi : \mathcal{X} \times \mathcal{Y} \longrightarrow \mathcal{Z}$ and to learn a value function $f : \mathcal{Z} \longrightarrow \mathbb{R}$; here, $\mathcal{Y}$ is a parametric or structured space of alternatives and $\mathcal{Z} \subseteq \mathbb{R}^d$ a joint feature space (Tsochantaridis et al. 2005; Schäfer and Hüllermeier 2015).

Learning Utility Functions

Evaluating alternatives in terms of a value or utility function is a very natural way of representing preferences, which has a long tradition in economics and decision theory (Fishburn 1969). In the object preferences scenario, such a function is a mapping $f : \mathcal{X} \longrightarrow \mathbb{R}$ that assigns a utility degree $f(x)$ to each object x and, thereby, induces a linear order on $\mathcal{X}$. In the label preferences scenario, a utility function $f_i : \mathcal{X} \longrightarrow \mathcal{U}$ is needed for every label λ_i, $i = 1, \ldots, m$. Here, $f_i(x)$ is the utility assigned to alternative λ_i in the context x. To obtain a ranking for x, the alternatives are ordered according to their utility scores, i.e., a ranking $\succ_x$ is derived such that $\lambda_i \succ_x \lambda_j$ implies $f_i(x) \geq f_j(x)$.

If the training data offers the utility scores directly, preference learning essentially reduces to a standard ▸regression or an ordinal regression problem, depending on the underlying utility scale. This information can rarely be assumed, however. Instead, usually only constraints derived from comparative preference information of the form "this alternative should have a higher utility score than that alternative" are given. Thus, the challenge for the learner is to find a value function that is as much as possible in agreement with a set of such constraints.

For object ranking approaches, this idea has first been formalized by Tesauro (1989) under the name *comparison training*. He proposed a symmetric neural-network architecture that can be trained with representations of two states and a training signal that indicates which of the two states is preferable. The elegance of this approach comes from the property that one can replace the two symmetric components of the network with a single network, which can subsequently provide a real-valued evaluation of single states. Similar ideas have also been investigated for training other types of classifiers, in particular support vector machines. We already mentioned Joachims (2002) who analyzed "click-through data" in order to rank documents retrieved by a search engine according to their relevance. Earlier, Herbrich et al. (1998) proposed an algorithm for training SVMs from pairwise preference relations between objects.

For the case of label ranking, a corresponding method for learning the functions $f_i(\cdot)$, $i = 1, \ldots, m$, from training data has been proposed in the framework of *constraint classification*, which allows for reducing a label ranking to a single binary classification problem (Har-Peled et al. 2002). The learning method proposed in this work constructs two training examples, a positive and a negative one, for each given preference $\lambda_i \succ_x \lambda_j$, where the original N-dimensional training example (feature vector) x is mapped into an $(m \times N)$-dimensional space. In this space, the learner finds a linear model (hyperplane) f that separates the positive from the negative examples. Finally, the model f is "split" into m linear value functions $f_1, \ldots, f_m$, one for each label.

Learning Preference Relations

An alternative to learning latent utility functions consists of learning binary preference relations, which essentially amounts to reducing preference learning to binary classification. For object ranking, the pairwise approach has been pursued in Cohen et al. (1999). The authors propose to solve object ranking problems by learning a binary preference predicate $Q(x, x')$, which predicts whether x is preferred to x' or vice versa. A final ordering is found in a second phase by deriving a ranking that is maximally consistent with these (possibly conflicting) predictions.

For label ranking, the pairwise approach has been introduced in Hüllermeier et al. (2008) as a natural extension of *pairwise classification*, a well-known ▸class binarization technique. The idea is to train a separate model (base learner) $\mathcal{M}_{i,j}$ for each pair of labels $(\lambda_i, \lambda_j) \in \mathcal{L}$, $1 \leq i < j \leq m$; thus, a total number of $m(m-1)/2$ models are needed. For training, a preference information of the form $\lambda_i \succ_x \lambda_j$ is turned into a (classification) example (x, y) for the learner $\mathcal{M}_{a,b}$, where $a = \min(i, j)$ and $b = \max(i, j)$. Moreover, $y = 1$ if $i < j$ and $y = 0$ otherwise. Thus, $\mathcal{M}_{a,b}$ is intended to learn the mapping that outputs 1 if $\lambda_a \succ_x \lambda_b$ and 0 if $\lambda_b \succ_x \lambda_a$. This mapping can be realized by any binary classifier. Instead of a $\{0, 1\}$-valued classifier, one can of course also employ a scoring classifier. For example, the output of a probabilistic classifier would be a number in the unit interval [0, 1] that can be interpreted as a probability of the preference $\lambda_a \succ_x \lambda_b$.

At classification time, a query $x_0 \in \mathcal{X}$ is submitted to the complete ensemble of binary learners. Thus, a collection of predicted pairwise preference degrees $\mathcal{M}_{i,j}(x)$, $1 \leq i, j \leq m$, is obtained. The problem, then, is to turn these pairwise preferences into a ranking of the label set $\mathcal{L}$. To this end, different ranking procedures can be used. The simplest approach is to extend the (weighted) voting procedure that is often applied in pairwise classification: For each label λ_i, a score

$$S_i = \sum_{1 \leq j \neq i \leq m} \mathcal{M}_{i,j}(x_0)$$

is derived (where $\mathcal{M}_{i,j}(x_0) = 1 - \mathcal{M}_{j,i}(x_0)$ for $i > j$), and then the labels are ordered according to these scores. Despite its simplicity, this ranking procedure has several appealing properties. Apart from its computational efficiency, it turned out to be relatively robust in practice, and, moreover, it possesses some provable optimality properties in the case where Spearman's rank correlation is used as an underlying accuracy measure. Roughly speaking, if the binary learners are unbiased probabilistic classifiers, the simple "ranking by weighted voting" procedure yields a label ranking that maximizes the expected Spearman rank correlation (Hüllermeier and Fürnkranz 2010). Finally, it is worth mentioning that, by changing the ranking procedure, the pairwise approach can also be adjusted to accuracy measures other than Spearman's rank correlation.

Other Approaches

Referring to the type of training data and the loss function to be minimized on this data, learning value functions and learning preference relations are sometimes called the "pointwise" and "pairwise" approach to preference learning, respectively. This is distinguished from the "listwise" approach, in which a loss is defined on a predicted ranking directly. This can be done, for example, on the basis of probabilistic models of ranking data, such as the Plackett-Luce model. The idea, then, is to learn the parameters of a probabilistic model using statistical methods such as maximum likelihood estimation (or, equivalently, minimizing logarithmic loss). Methods of this kind have been proposed both for object ranking (Cao et al. 2007) and label ranking (Cheng et al. 2010).

Yet another alternative is to resort to the idea of local estimation techniques as prominently represented, for example, by the ▶ nearest neighbor estimation principle: Considering the rankings observed in similar situations as representative, a ranking for the current situation is estimated on the basis of these neighbor rankings, namely, by finding a suitable consensus among them; essentially, this is a problem of rank aggregation (Cheng et al. 2009).

Future Directions

As already said, preference learning is an emerging branch of machine learning and still developing quite dynamically. In particular, new settings or variants of existing frameworks will certainly be proposed and studied in the future. As for ranking problems, for example, an obvious idea and reasonable extension is to go beyond strict total order relations and instead allow for *incomparability* or *indifference* between alternatives and for representing uncertainty about predicted relations (Cheng et al. 2012). Another interesting direction is to combine preference learning with ▶ online learning, i.e., to predict preferences in an online setting. First steps in the direction of online preference learning have recently been made with a preference-based variant of the ▶ multiarmed bandit problem (Busa-Fekete and Hüllermeier 2014).

Cross-References

- ▶ Class Binarization
- ▶ Classification
- ▶ Metalearning

- ▶ Multi-armed bandit
- ▶ Online Learning
- ▶ Rank Correlation
- ▶ Regression

Recommended Reading

Boutilier C, Brafman R, Domshlak C, Hoos H, Poole D (2004) CP-nets: a tool for representing and reasoning with conditional ceteris paribus preference statements. J AI Res 21:135–191

Busa-Fekete R, Hüllermeier E (2014) A survey of preference-based online learning with bandit algorithms. In: Proceedings of ALT, 25th international conference on algorithmic learning theory, Bled. Springer, pp 18–39

Cao Z, Qin T, Liu TY, Tsai MF, Li H (2007) Learning to rank: from pairwise approach to listwise approach. In: Proceedings of ICML, 24th international conference on machine learning, pp 129–136

Cheng W, Hühn J, Hüllermeier E (2009) Decision tree and instance-based learning for label ranking. In: Proceedings of ICML–2009, 26th international conference on machine learning, Montreal, pp 161–168

Cheng W, Dembczynski K, Hüllermeier E (2010) Label ranking based on the Plackett-Luce model. In: Proceedings of ICML–2010, international conference on machine learning, Haifa, pp 215–222

Cheng W, Hüllermeier E, Waegeman W, Welker V (2012) Label ranking with partial abstention based on thresholded probabilistic models. In: Proceedings of NIPS–2012, 26th annual conference on neural information processing systems, Lake Tahoe

Cohen WW, Schapire RE, Singer Y (1999) Learning to order things. J Artif Intell Res 10(1):243–270

Domshlak C, Hüllermeier E, Kaci S, Prade H (2011) Preferences in AI: an overview. Artif Intell 175(7–8):1037–1052

Fishburn PC (1969) Utility-theory for decision making. Wiley, New York

Fürnkranz J, Hüllermeier E (eds) (2010) Preference learning. Springer, Heidelberg/New York

Fürnkranz J, Hüllermeier E (2010) Preference learning: an introduction. In: Preference learning. Springer, Heidelberg/New York, pp 1–18

Fürnkranz J, Hüllermeier E, Vanderlooy S (2009) Binary decomposition methods for multipartite ranking. In: Proceedings of ECML/PKDD–2009, European conference on machine learning and knowledge discovery in databases, Bled

Har-Peled S, Roth D, Zimak D (2002) Constraint classification: a new approach to multiclass classification. In: Proceedings of 13th international conference on algorithmic learning theory, Lübeck. Springer, pp 365–379

Herbrich R, Graepel T, Bollmann-Sdorra P, Obermayer K (1998) Supervised learning of preference relations. In: Proceedings des Fachgruppentreffens Maschinelles Lernen (FGML-98), pp 43–47

Hüllermeier E, Fürnkranz J (2010) On predictive accuracy and risk minimization in pairwise label ranking. J Comput Syst Sci 76(1):49–62

Hüllermeier E, Fürnkranz J, Cheng W, Brinker K (2008) Label ranking by learning pairwise preferences. Artif Intell 172:1897–1917

Joachims T (2002) Optimizing search engines using clickthrough data. In: Proceedings of KDD–02, 8th ACM SIGKDD international conference on knowledge discovery and data mining. ACM Press, pp 133–142

Kamishima T, Kazawa H, Akaho S (2010) A survey and empirical comparison of object ranking methods. In: Fürnkranz J, Hüllermeier E (eds) Preference learning. Springer, Heidelberg/New York, pp 181–202

Liu TY (2011) Learning to rank for information retrieval. Springer, Berlin/Heidelberg/New York

Schäfer D, Hüllermeier E (2015) Dyad ranking using a bilinear Plackett-Luce model. In: Proceedings of ECML/PKDD–2015, European conference on machine learning and knowledge discovery in databases, Porto

Tesauro G (1989) Connectionist learning of expert preferences by comparison training. In: Advances in neural information processing systems 1 (NIPS-88). Morgan Kaufmann, pp 99–106

Tsochantaridis I, Joachims T, Hofmann T, Altun Y (2005) Large margin methods for structured and interdependent output variables. J Mach Learn Res 6:1453–1484

Vembu S, Gärtner T (2010) Label ranking: a survey. In: Fürnkranz J, Hüllermeier E (eds) Preference learning. Springer, Heidelberg/New York

Zhou Y, Lui Y, Yang J, He X, Liu L (2014) A taxonomy of label ranking algorithms. J Comput 9(3):557

Pre-pruning

Synonyms

Stopping criteria

Definition

Pre-pruning is a ▶ Pruning mechanism that monitors the learning process and prevents further refinements if the current hypothesis becomes too complex.

Cross-References

- ▶ Overfitting
- ▶ Post-pruning
- ▶ Pruning

Presynaptic Neuron

The neuron that sends signals across a synaptic connection. A chemical synaptic connection between two neurons allows to transmit signals from a presynaptic neuron to a postsynaptic neuron.

Principal Component Analysis

Synonyms

PCA

Definition

Principal Component Analysis (PCA) is a ▶ dimensionality reduction technique. It is described in ▶ covariance matrix.

Prior

▶ Prior Probability

Prior Probability

Geoffrey I. Webb
Faculty of Information Technology, Monash University, Victoria, Australia

Synonyms

Prior

Definition

In Bayesian inference, a *prior probability* of a value x of a random variable X, $\mathrm{P}(X = x)$, is the probability of X assuming the value x in the absence of (or before obtaining) any additional information. It contrasts with the ▶ posterior probability, $\mathrm{P}(X = x|Y = y)$, the probability of X assuming the value x in the context of $Y = y$.

For example, it may be that the prevalence of a particular form of cancer, *exoma*, in the population is 0.1 %, so the prior probability of exoma, P(exoma = true), is 0.001. However, assume 50 % of people who have skin discolorations of greater than 1 cm width (sd > 1 cm) have exoma. It follows that the posterior probability of exoma given sd > 1 cm, P(exoma = true | sd > 1 cm = true), is 0.500.

Cross-References

- ▶ Bayesian Methods

Privacy-Preserving Data Mining

▶ Privacy-Related Aspects and Techniques

Privacy-Related Aspects and Techniques

Stan Matwin
University of Ottawa, Ottawa, ON, Canada
Polish Academy of Sciences, Warsaw, Poland

Synonyms

Privacy-preserving data mining

Definition

The privacy-preserving aspects and techniques of machine learning cover the family of methods and architectures developed to protect the privacy of people whose data are used by machine learning (ML) algorithms. This field, also known as privacy-preserving data mining (PPDM), addresses the issues of data privacy in ML and data mining. Most existing methods and approaches are intended to hide the original data from the learning algorithm, while there is emerging interest in methods ensuring that the learned model does not reveal private information. Another research direction contemplates methods in which several parties bring their data into the model-building process without mutually revealing their own data.

Motivation and Background

The key concept for any discussion of the privacy aspects of data mining is the definition of privacy. After Alan Westin, we understand privacy as the ability "of individuals . . . to determine for themselves when, how, and to what extent information about them is communicated to others" (Westin 1967). One of the main societal concerns about modern computing is that the storing, keeping, and processing of massive amounts of data may jeopardize the privacy of individuals whom the data represent. In particular, ML and its power to find patterns and infer new facts from existing data makes it difficult for people to control information about themselves. Moreover, the infrastructure normally put together to conduct large-scale model building (e.g., large data repositories and data warehouses), is conducive to misuse of data. Personal data, amassed in large collections that are easily accessed through databases and often available online to the entire world, become – as phrased by Moor in an apt metaphor (Moor 2004) – "greased." It is difficult for people to control the use of this data.

Theory/Solutions

Basic Dimensions of Privacy Techniques

Privacy-related techniques can be characterized by: (1) the kind of source data modification they perform, e.g., data perturbation, randomization, generalization, and hiding; (2) the ML algorithm that works on the data and how is it modified to meet the privacy requirements imposed on it; and (3) whether the data are centralized or distributed among several parties, and – in the latter case – on what the distribution is based. But even at a more basic level, it is useful to view privacy-related techniques along just two fundamental dimensions.

The first dimension defines what is protected as private – is it the data itself, or the model (the results of data mining)? As we show below, the knowledge of the latter can also lead to identifying and revealing information about individuals. The second dimension defines the protocol of the use of the data: are the data centralized and owned by a single owner, or are the data distributed among multiple parties? In the former case, the owner needs to protect the data from revealing information about individuals represented in the data when that data is being used to build a model by someone else. In the latter case, we assume that the parties have limited trust in each other: they are interested in the results of data mining performed on the union of the data of all the parties, while not trusting the other parties to see their own data without first protecting it against disclosure of information about individuals to other parties.

Moreover, work in PPDM has to apply a framework that is broader than the standard ML methodology. When privacy is an important goal, what matters in performance evaluation is not only the standard ML performance measures, but also some measure of the privacy achieved, as well as some analysis of the robustness of the approach to attacks.

In this article, we structure our discussion of the current work on PPDM in terms of the taxonomy proposed above. This leads to the following bird's-eye view of the field.

Protecting Centralized Data

This subfield emerged in 2000 with the seminal paper by Agrawal and Srikant (2000). They stated the problem as follows: given data in the standard ▸ attribute-value representation, how can an accurate ▸ decision tree be built so that, instead of using original attribute values x_i, the decision tree induction algorithm takes input values $x_i + r$, where r belongs to a certain distribution (Gaussian or uniform). This is a data perturbation technique: the original values are changed beyond recognition, while the distributional properties of the entire data set that decision tree ▸ induction uses remain the same, at least up to a small (empirically, less than 5 %) degradation in accuracy. There is a clear trade-off between the privacy assured by this approach and the quality of the model compared to the model obtained from the original data. This line of research has been continued in Evfimievski et al. (2002) where the approach is extended to association rule mining. As a note of caution about these results, Kargupta et al. (2003) have shown, in 2003, how the randomization approaches are sensitive to attack. They demonstrate how the noise that randomly perturbs the data can be viewed as a random matrix, and that the original data can be accurately estimated from the perturbed data using a spectral filter that exploits some theoretical properties of random matrices.

The simplest and most widely used privacy preservation technique is anonymization of data (also called de-identification). In the context of de-identification, it is useful to distinguish three types of attributes.

Explicit identifiers allow direct linking of an instance to a person (e.g., a cellular phone number or a driver's license number to its holder).

Quasi-identifiers, possibly combined with other attributes, may lead to other data sources capable of unique identification. For instance, Sweeney (2001) shows that the quasi-identifier triplet <date of birth, 5 digit postal code, gender>, combined with the voters' list (publicly available in the USA) uniquely identifies 87 % of the population of the country. As a convincing application of this observation, using quasi-identifiers, Sweeney was able to obtain health records of the governor of Massachusetts from a published dataset of health records of all state employees in which only explicit identifiers have been removed.

Finally, non-identifying attributes are those for which there is no known inference linking to an explicit identifier. Usually performed as part of data preparation, anonymization removes all explicit identifiers from the data.

While anonymization is by far the most common privacy-preserving technique used in practice, it is also the most fallible one. In August 2006, for the benefit of the Web Mining Research community, AOL published 20 million search records (queries and URLs the members had visited) from 658,000 of its members. AOL had performed what it believed was anonymization, in the sense that it removed the names of the members. However, based on the queries – which often contained information that would identify a small set of members or a unique person – it was easy, in many cases, to manually re-identify the AOL member using secondary public knowledge sources. An inquisitive New York Times journalist identified one member and interviewed her.

L. Sweeney is to be credited with sensitizing the privacy community to the fallacy of anonymization: "Shockingly, there remains a common incorrect belief that if the data look anonymous, it is anonymous" (Sweeney 2001). Even if information is de-identified today, future data sources may make re-identification possible. As anonymization is very commonly used prior to model building from medical data, it is interesting that this type of data is prone to specific kinds of re-identification, and therefore anonymization of medical data should be done with particular skill and understanding of the data. Malin (2005) shows how the four main de-identification techniques used in anonymization of genomic data are prone to known, published attacks that can re-identify the data. Moreover, he points out that there will never be certainty about de-identification for quasi-identifiers, as new attributes and data sources that can lead to a linkage to explicitly identifying attributes are constantly being engineered as part of genetics research.

Other perturbation approaches targeting binary data involve changing (flipping) values of selected attributes with a given probability (Du and Zhan 2003; Zhan and Matwin 2004), or replacing the original attribute with a value that is more general in some pre-agreed taxonomy (Iyengar 2002). Generalization approaches often use the concept of k-anonymity: any instance in the database is indistinguishable from other $k-1$ instances (for every row in the database there are $k-1$ identical rows). Finding the least general k-anonymous generalization of a database (i.e., moving the least number of edges upward in a given taxonomy) is an optimization task, known to be *NP*-complete. There are heuristic solutions proposed for it; e.g., Iyengar (2002) uses a ▸ genetic algorithm for this task. Friedman et al. (2006) shows how to build k-anonymity into the decision tree induction. Lately, PPDM researchers have pointed out some weaknesses of the k-anonymity approach. In particular, attacks on data with some properties (e.g., skewed distribution of values of a sensitive attribute, or specific background knowledge) have been described, and techniques to prevent such attacks have been proposed. The notion of p-sensitivity or l-diversity proposed in Machanavajjhala et al. (2007) addresses these weaknesses of k-anonymity by modifying k-anonymity techniques so that the abovementioned attacks do not apply. Furthermore, t-closeness (Ninghui et al. 2007) shows certain shortcomings of these techniques and the resulting potential attacks, and proposes a data perturbation technique which ensures that the distribution of the values of the sensitive attribute in any group resulting from anonymization is close to its distribution in the original table. Some authors, e.g., Domingo-Ferrer et al. (2008), propose the integration of several techniques addressing shortcomings of k-anonymity into a single perturbation technique. The drawback of these solutions is that they decrease the utility of the data more than the standard k-anonymity approaches.

Protecting the Model (Centralized Data)

Is it true that when the data are private, there will be no violation of privacy? The answer is no. In some circumstances, the model may reveal private information about individuals. Atzori et al. (2005) gives an example of such a situation for association rules: suppose the ▸ association rule $a_1 \wedge a_2 \wedge a_3 \Rightarrow a_4$ has support sup = 80, confidence conf = 98.7 %. This rule is 80-anonymous, but considering that

$$\begin{aligned}\sup(\{a_1, a_2, a_3\}) &= \frac{\sup(\{a_1, a_2, a_3, a_4\})}{\text{conf}} \\ &= \frac{80}{0.0987} \approx 81.05\end{aligned}$$

and given that the pattern $a_1 \wedge a_2 \wedge a_3 \wedge a_4$ holds for 80 individuals, and the pattern $a_1 \wedge a_2 \wedge a_3$ holds for 81 individuals, clearly the pattern $a_1 \wedge a_2 \wedge a_3 \wedge \neg a_4$ holds for just one person. Therefore, the rule unexpectedly reveals private information about a specific person. Atzori et al. (2005) proposes to apply k-anonymity to patterns instead of data, as in the previous section. The authors define inference channels as ▸ itemsets from which it is possible to infer other itemsets that are not k-anonymous, as in the above example. They then show an efficient way to represent and compute inference channels, which, once known, can be blocked from the output of an association rule finder. The inference channel problem is also discussed in Oliveira et al. (2004), where itemset "sanitization" removes itemsets that lead to sensitive (non-k-anonymous) rules.

This approach is an interesting continuation of Sweeney's classical work (Sweeney 2001), and it addresses an important threat to privacy ignored by most other approaches based on data perturbation or cryptographic protection of the data.

Distributed Data

Most of the work mentioned above addresses the case of centralized data. The distributed situation, however, is often encountered and has important applications. Consider, for example, several hospitals involved in a multi-site medical trial that want to mine the data describing the union of their patients. This increases the size of the population subject to data analysis, thereby increasing the scope and the importance of the trial. In another example, a car manufacturer performing data analysis on the set of vehicles exhibiting a given problem wants to represent data about different components of the vehicle originating in databases of the suppliers of these components. In general, if we abstractly represent the database as a table, there are two collaborative frameworks in which data is distributed. Horizontally partitioned data is distributed by rows (all parties have the same attributes, but different instances – as in the medical study example). Vertically partitioned data is distributed by columns (all parties have the same instances; some attributes belong to specific parties, and some, such as the class, are shared among all parties – as in the vehicle data analysis example).

An important branch of research on learning from distributed data while parties do not reveal their data to each other is based on results from computer security, specifically from cryptography and from the secure multiparty computation (SMC). Particularly interesting is the case when there is no trusted external party – all the computation is distributed among parties that collectively hold the partitioned data. SMC has produced constructive results showing how any Boolean function can be computed from inputs belonging to different parties, so that the parties never get to know input values that do not belong to them. These results are based on the idea of splitting a single data value between two parties into "shares," so that none of them knows the value but they can still do computation on the shares using a gate such as *exclusive or* Yao (1986). In particular, there is an SMC result known as secure sum: k parties have private values x_i and they want to compute $\sigma_I x_i$ without disclosing their x_i to any other party. This result, and similar results for value comparison and other simple functions, are the building blocks of many privacy-preserving ML algorithms. On that basis, a number of standard ▸ classifier induction algorithms, in their horizontal and vertical partitioning versions, have been published, including decision tree (ID3) induction (Friedman et al. 2006), Naïve Bayes, the ▸ Apriori association rule mining algorithm (Kantarcioglu and Clifton 2004; Vaidya and Clifton 2002), and many others.

We can observe that data privacy issues extend to the use of the learned model. For horizontal partitioning, each party can be given the model and apply it to the new data. For vertical partitioning, however, the situation is more difficult: the parties, all knowing the model, have to compute their part of the decision that the model delivers, and have to communicate with selected other parties after this is done. For instance, for decision trees, a node n applies its test and contacts the party holding the attribute in the child c chosen by the test, giving c the test to perform. In this manner, a single party n only knows the result of its test (the corresponding attribute value) and the tests of its children (but not their outcomes). This is repeated recursively until the leaf node is reached and the decision is communicated to all parties.

A different approach involving cryptographic tools other than Yao's circuits is based on the concept of homomorphic encryption (Paillier 1999). Encryption e is homomorphic with respect to some operation * in the message space if there is a corresponding operation $^{*\prime}$ in the ciphertext space, such that for any messages m1, m2, e(m1)$^{*\prime}$ e(m2) $=$ e(m1*m2). The standard RSA encryption is homomorphic with $^{*\prime}$ being logical multiplication and * logical addition on sequences of bytes. To give a flavor of the use of homomorphic encryption, let us see in detail how this kind of encryption is used in computing the scalar product of two binary vectors.

Assume just two parties, Alice and Bob. They both have their private binary vectors A$_{1,\ldots N}$, B$_{1,\ldots,N}$. In association rule mining, A$_i$ and B$_i$ represent A's and B's transactions projected on the set of items whose frequency is being computed. In our protocol, one of the parties is randomly chosen as a key generator. Assume Alice is selected as the key generator. Alice generates an encryption key (e) and a decryption key (d). She applies the encryption key to the sum of each value of A and a digital envelope $R_i{}^*X$ of A_i (i.e., $e(A_i i + R_i{}^*X)$), where R_i is a random integer and X is an integer that is greater than N. She then sends $e(A_i + R_i{}^*X)$s to Bob. Bob computes the multiplication $M = \prod_{j=1}^{N} [e(A_j + R_i{}^*X) \times B_j]$ when $B_j = 1$ (as when $B_j = 0$, the result of multiplication does not contribute to the frequency count). Now, $M = e(A_1 + A_2 + \cdots + A_j + (R_1 + R_2 + \cdots + R_1)^*X)$ due to the property of homomorphic encryption. Bob sends

Privacy-Related Aspects and Techniques, Table 1 Classification taxonomy to systematize the discussion of the current work in PPDM

	Data centralized	Data distributed
Protecting the data	Agrawal and Srikant (2000), Evfimievski et al. (2002), Du and Zhan (2003), and Iyengar (2002)	Vaidya and Clifton (2002), Vaidya et al. (2008), and Kantarcioglu and Clifton (2004)
Protecting the model	Oliveira et al. (2004), Atzori et al. (2005), Felty and Matwin (2002), and Friedman et al. (2006)	Jiang and Atzori (2006)

the result of this multiplication to Alice, who computes $[d(e(A_1 + A_2 + \cdots + A_j + (R_1 + R_2 + \cdots + R_1)^* X)]) \bmod X = (A_1 + A_2 + \cdots + A_1 + (R_1 + R_2 + \cdots + R_j)^* X) \bmod X$ and obtains the scalar product. This scalar product is directly used in computing the frequency count of an itemset, where N is the number of items in the itemset, and A_i, B_i are Alice's and Bob's transactions projected on the itemset whose frequency is computed.

While more efficient than the SMC-based approaches, homomorphic encryption methods are more prone to attack, as their security is based on a weaker security concept (Paillier 1999) than Yao's approach. In general, cryptographic solutions have the advantage of protecting the source data while leaving it unchanged: unlike data modification methods, they have no negative impact on the quality of the learned model. However, they have a considerable cost impact in terms of complexity of the algorithms, computation cost of the cryptographic processes involved, and the communication cost for the transmission of partial computational results between the parties (Subramaniam et al. 2004). Their practical applicability on real-life-sized datasets still needs to be demonstrated.

The discussion above focuses on protecting the data. In terms of our diagram in Table 1, we have to address its right column. Here, methods have been proposed to mainly address mainly the north-east entry of the diagram. In particular, in Vaidya and Clifton (2002) propose a method to compute association rules in an environment where data is distributed. In particular, their method addresses the case of vertically partitioned data, where different parties hold different attribute sets for the same instances. The problem is solved without the existence of a trusted third party, using SMC. Independently, we have obtained a different solution to this task using homomorphic encryption techniques (Zhan et al. 2007). Many papers have presented solutions for both vertically and horizontally partitioned data, and for different data mining tasks, e.g., Friedman et al. (2006) and Vaidya et al. (2006).

Moreover, Jiang and Atzori (2006) have obtained a solution for the model-protection case in a distributed setting (south-east quadrant in Table 1). Their work is based on a cryptographic technique, and addresses the case of vertical partitioning of the data among parties.

Evaluation

The evaluation of privacy-related techniques must be broader than standard ML evaluation. Besides evaluating the performance of the ML component using the appropriate tool (e.g., ▸ accuracy, ▸ ROC, support/confidence), one also needs to evaluate the various privacy aspects of a learned model. This is difficult, as there is no commonly accepted definition of privacy. Even if there were one, it would not be in quantitative, operational terms that can be objectively measured, but most certainly with references to moral and social values. For instance, Clifton (2005) points out that a definition of privacy as the "freedom from unauthorized intrusion" implies that we need to understand what constitutes an intrusion and that we can measure its extent. For these reasons, most definitions in current privacy-preserving data mining research are method-specific, without any comparison between different methods. For example, the classic work of Agrawal and Srikant (2000) measures privacy after data perturbation as the size of the interval to which the original value can be estimated. If we know that the original value was 0.5, and following a perturbation its best estimate is, with 95 % confidence, within the interval [0.3, 0.7], then the amount of privacy is the size of this interval, (i.e., 0.4, with a confidence of 95 %). Later, Agrawal and Aggarwal (2001) proposed a more general measure of data privacy measuring this property of a dataset that has been subject to one of the data perturbation techniques. The idea is that if noise from a random variable A is added to the data, we can measure the uncertainty of the perturbed values using differential entropy inherent in A. Specifically, if we add noise from a random variable A, the privacy is

$$\prod(A) = 2^{-f_{\Omega A}^{fA(a)\log_2 f_A(a)da}},$$

where Ω_A is the domain of A. Privacy is 0 if the exact value is known (the entropy is ∞); if it is known that the data is in the interval of length $a, \prod(A) = a$.

Clifton (2005) argues that if disclosure is only possible to a group of people rather than a single person, then the size of the group is a natural measure of privacy. This is the case for k-anonymity methods. He further argues that a good evaluation measure should not only capture the likelihood of linking an ML result to an individual, but should also capture how intrusive this linking is. For instance, an association rule with a support value of 50 and a confidence level of 100 % is 50-anonymous, but it also reveals the consequent of the rule to all 50 participants.

Finally, the style of evaluation needs to take into account attack analysis, as in Malin (2005).

Future Directions

One of the most pressing challenges for the community is to work out a quantifiable and socially comprehensible definition of privacy for the purpose of privacy-preserving techniques. This is clearly a difficult problem, likely not solvable by ML or even computer science alone. As privacy has basic social and economic dimensions, economics may contribute to an acceptable definition, as already explored in Rossi (2004).

Another important question is the ability to analyze data privacy, including inference from data using ML, in the context of specific rules and regulations, e.g., HIPAA (Health and Services 2003) or the European Privacy Directive (1995). First forays in this direction using formal methods have already been made, e.g., Barth et al. (2006) and Felty and Matwin (2002).

Finally, the increasing abundance and availability of data tracking mobile devices will bring new challenges to the field. People will become potentially identifiable by knowing the trajectories their mobile devices leave in fixed times and time intervals. Clearly such data, already collected, present an important asset from the public security point of view, but also a very considerable threat from a privacy perspective. There is early work in this area (Gianotti and Pedreschi 2008). Such data are already being collected. This is an important asset for public security, but also a considerable threat for privacy.

Recommended Reading

Agrawal D, Aggarwal CC (2001) On the design and quantification of privacy preserving data mining algorithms. In: Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems. ACM, Santa Barbara

Agrawal R, Srikant R (2000) Privacy-preserving data mining. ACM SIGMOD Rec. 29(Part 2):439–450

Atzori M, Bonchi F, Giannotti F, Pedreschi D (2005) k-Anonymous patterns. In: Proceedings of the ninth European conference on principles and practice of knowledge discovery in databases (PKDD 05), Porto

Barth A, Datta A, Mitchell JC, Nissenbaum H (2006) Privacy and contextual integrity: framework and applications. IEEE Symp Secur Priv 184–198

Clifton CW (2005) What is privacy? Critical steps for privacy-preserving data mining, workshop on privacy and security aspects of data mining

Directive (1995) Directive 95/46/EC of the European Parliament on the protection of individuals with regard to the processing of personal data and on the free movement of such data. Off J Eur Commun 38(L281):0031–0050

Domingo-Ferrer J, Sebé F, Solanas A (2008) An anonymity model achievable via microaggregation. In: VLDB workshop on secure data management, Auckland. Springer, pp 209–218

Du W, Zhan Z (2003) Using randomized response techniques for privacy-preserving data mining. In: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining, Washington, DC, vol 510

Evfimievski A, Srikant R, Agrawal R, Gehrke J (2002) Privacy preserving mining of association rules. In: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining, Edmonton, pp 217–228

Felty A, Matwin S (2002) Privacy-oriented data mining by proof checking. In: Sixth European conference on principles of data mining and knowledge discovery, Helsink, vol 2431, pp 138–149

Friedman A, Schuster A, Wolff R (2006) k-anonymous decision tree induction. In: PKDD 2006, Berlin, pp 151–162

Health UDo, Services H (eds) (2003) Summary of HIPAA privacy rule. US Department of Health and Human Services, Washington, DC

Gianotti F, Pedreschi D (2008) Mobility, data mining and privacy: geographic knowledge discovery. Springer, Berlin

Iyengar VS (2002) Transforming data to satisfy privacy constraints. In: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining, Edmonton, pp 279–288

Jiang W, Atzori M (2006) Secure distributed k-Anonymous pattern mining. In: Proceedings of the sixth international conference on data mining, Hong Kong. IEEE Computer Society

Kantarcioglu M, Clifton C (2004) Privacy-preserving distributed mining of association rules on horizontally partitioned data. IEEE Trans Knowl Data Eng 16:1026–1037

Kargupta H, Datta S, Wang Q (2003) On the privacy preserving properties of random data perturbation techniques. In: Third IEEE international conference on data mining (ICDM 2003), Melbourne, pp 99–106

Machanavajjhala A, Kifer D, Gehrke J, Venkitasubramaniam M (2007) L-diversity: privacy beyond k-anonymity. ACM Trans Knowl Discov Data 1:3

Malin BA (2005) An evaluation of the current state of genomic data privacy protection technology and a roadmap for the future. J Am Med Inf Assoc 12:28

Moor J (2004) Towards a theory of privacy in the information age. In: Bynum T, Rodgerson S (eds) Computer ethics and professional responsibility. Blackwell, Malden

Ninghui L, Tiancheng L, Venkatasubramanian S (2007) t-closeness: privacy beyond k-anonymity and l-diversity. In: IEEE 23rd international conference on data engineering (ICDE 2007), Istanbul, pp 106–115

Oliveira SRM, Zaïane OR, Saygin Y (2004) Secure association rule sharing. In: Proceedings of the eighth PAKDD and advances in knowledge discovery and data mining, Sydney, pp 74–850

Paillier P (1999) The 26th international conference on privacy and personal data protection, advances in cryptography (EUROCRYPT'99), Prague, pp 23–38

Rossi G (2004) Privacy as quality in modern economy. In: The 26th international conference on privacy and personal data protection, Wroclaw

Subramaniam H, Wright RN, Yang Z (2004) Experimental analysis of privacy-preserving statistics computation. In: Proceedings of the VLDB workshop on secure data management, Toronto, pp 55–66

Sweeney L (2001) Computational disclosure control: a primer on data privacy protection. Massachusetts Institute of Technology, Deptartment of Electrical Engineering and Computer Science, Cambridge

Vaidya J, Clifton C (2002) Privacy preserving association rule mining in vertically partitioned data. In: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, Edmonton, pp 639–644

Vaidya J, Clifton C, Kantarcioglu M, Patterson AS (2008) Privacy-preserving decision trees over vertically partitioned data. ACM Trans Knowl Discov Data 2:1–27

Vaidya J, Zhu YM, Clifton CW (2006) Privacy preserving data mining. Springer, New York

Website of the GeoPKDD Project (2006)

Westin A (1967) Privacy and freedom. Atheneum, New York

Yao A (1986) How to generate and exchange secrets. In: 27th FOCS, Toronto

Zhan J, Matwin S, Chang L (2007) Privacy-preserving collaborative association rule mining. J Netw Comput Appl 30:1216–1227

Zhan JZ, Matwin S (2004) Privacy-prteserving data mining in electronic surveys. In: ICEB 2004, Beijing, pp 1179–1185

Probabilistic Context-Free Grammars

Yasubumi Sakakibara
Keio University, Hiyoshi, Kohoku-ku, Japan

Synonyms

PCFG

Definition

In formal language theory, formal grammar (phrase-structure grammar) is developed to capture the generative process of languages (Hopcroft and Ullman 1979). A formal grammar is a set of productions (rewriting rules) that are used to generate a set of strings, that is, a *language*. The productions are applied iteratively to generate a string, a process called *derivation*. The simplest kind of formal grammar is a *regular* grammar.

Context-free grammars (CFG) form a more powerful class of formal grammars than regular grammars and are often used to define the syntax of programming languages. Formally, a CFG consists of a set of nonterminal symbols N, a terminal alphabet Σ, a set P of productions (rewriting rules), and a special nonterminal S called the start symbol. For a nonempty set X of symbols, let X^* denote the set of all finite strings of symbols in X. Every CFG production has the form $S \rightarrow \alpha$, where $S \in N$ and $\alpha \in (N \cup \Sigma)^*$. That is, the left-hand side consists of one nonterminal and there is no restriction on the number or placement of nonterminals and terminals on the right-hand side. The *language generated* by a CFG G is denoted $L(G)$.

A *probabilistic context-free grammar* (PCFG) is obtained by specifying a probability for each production for a nonterminal A in a CFG, such that a probability distribution exists over the set of productions for A.

A CFG $G = (N, \Sigma, P, S)$ is in *Chomsky normal form* if each production rule is of the form $A \rightarrow BC$ or $A \rightarrow a$, where $A, B, C \in N$ and $a \in \Sigma$.

Given a PCFG G and a string $w = a_1 \ldots a_m$, there are three basic problems:

1. Calculating the probability $\Pr(w|G)$ that the grammar G assigns to w
2. Finding the most likely derivation (parse tree) of w by G
3. Estimating the parameters of G to maximize $\Pr(w|G)$

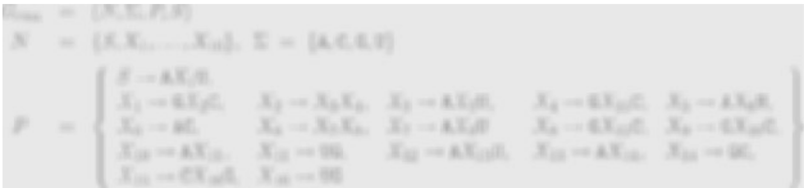
The first two problems, calculating the probability $\Pr(w|G)$ of a given string w assigned by a PCFG G and finding the most likely derivation of w by G, can be solved using dynamic programming methods analogous to the Cocke-Younger-Kasami or Early parsing methods. A polynomial-time algorithm for solving the second problem is known as *Viterbi* algorithm, and a polynomial-time algorithm for the third problem is known as the *inside-outside* algorithm (Lari and Young 1990).

Derivation Process

A *derivation* is a rewriting of a string in $(N \cup \Sigma)^*$ using the production rules of a CFG G. In each step of the derivation, a nonterminal from the current string is chosen and replaced with the right-hand side of a production rule for that nonterminal. This replacement process is repeated until the string consists of terminal symbols only. If a derivation begins with a nonterminal A and derives a string $\alpha \in (N \cup \Sigma)^*$, we write $A \Rightarrow \alpha$.

For example, the grammar in Fig. 1 generates an RNA sequence AGAAACUUGCUGGCCU by the following derivation: Beginning with the start symbol S, any production with S left of the arrow can be chosen to replace S. If the production $S \rightarrow \mathrm{A}X_1\mathrm{U}$ is selected (in this case, this is the only production available), the effect is to replace S with $\mathrm{A}X_1\mathrm{U}$. This one derivation step is written $S \Rightarrow \mathrm{A}X_1\mathrm{U}$, where the double arrow signifies application of a production. Next, if the production $X_1 \rightarrow \mathrm{G}X_2\mathrm{C}$ is selected, the derivation step is $\mathrm{A}X_1\mathrm{U} \Rightarrow \mathrm{AG}X_2\mathrm{CU}$. Continuing with similar derivation operations, each time choosing a nonterminal symbol and replacing it with the right-hand side of an appropriate production, we obtain the following derivation terminating with the desired sequence:

$$\begin{aligned} S &\Rightarrow \mathrm{A}X_1\mathrm{U} \Rightarrow \mathrm{AG}X_2\mathrm{CU} \Rightarrow \mathrm{AG}X_3X_4\mathrm{CU} \\ &\Rightarrow \mathrm{AGA}X_5\mathrm{U}X_4\mathrm{CU} \Rightarrow \mathrm{AGAA}X_6\mathrm{UU}X_4\mathrm{CU} \\ &\Rightarrow \mathrm{AGAAACUU}X_4\mathrm{CU} \end{aligned}$$



Probabilistic Context-Free Grammars, Fig. 1 This set of productions P generates RNA sequences with a certain restricted structure. $S, X_1, \ldots, X_{16}$ are nonterminals; A, U, G, and C are terminals representing the four nucleotides. Note that only for X_6 is there a choice of productions

$$\Rightarrow \text{AGAAACUUG}X_{15}\text{CCU}$$
$$\Rightarrow \text{AGAAACUUGC}X_{16}\text{GCCU}$$
$$\Rightarrow \text{AGAAACUUGCUGGCCU}.$$

Such a derivation can be arranged in a tree structure called a *parse tree*.

The *language generated* by a CFG G is denoted $L(G)$, that is, $L(G) = \{w | S \Rightarrow w, w \in \Sigma^*\}$. Two CFGs G and G' are said to be *equivalent* if and only if $L(G) = L(G')$.

Probability Distribution

A PCFG assigns a probability to each string which it derives and hence defines a probability distribution on the set of strings. The probability of a derivation can be calculated as the product of the probabilities of the productions used to generate the string. The probability of a string w is the sum of probabilities over all possible derivations that could generate w, written as follows:

$$\Pr(w|G) = \sum_{\text{all derivations } d} \Pr(S \overset{d}{\Rightarrow} w|G)$$
$$= \sum_{\alpha_1,\ldots,\alpha_n} \Pr(S \Rightarrow \alpha_1|G) \cdot \Pr(\alpha_1 \Rightarrow \alpha_2|G)$$
$$\ldots \Pr(\alpha_n \Rightarrow w|G).$$

Parsing Algorithm

Efficiently computing the probability of a string w, $\Pr(s|G)$, presents a problem because the number of possible derivations for w is exponential in the length of the string. However, a dynamic programming technique analogous to the Cocke-Kasami-Young or Earley methods for nonprobabilistic CFGs can accomplish this task efficiently (in time proportional to the cube of the length of w).

The CYK algorithm is a polynomial time algorithm for solving the parsing (membership) problem of CFGs using dynamic programming. The CYK algorithm assumes Chomsky normal form of CFGs, and the essence of the algorithm is the construction of a triangular *parse table T*. Given a CFG $G = (N, \Sigma, P, S)$ and an input string $w = a_1 a_2 \ldots a_n$ in Σ^* to be parsed according to G, each element of T, denoted $t_{i,j}$, for $1 \leq i \leq n$ and $1 \leq j \leq n - i + 1$, has a value which is a subset of N. The interpretation of T is that a nonterminal A is in $t_{i,j}$ if and only if $A \Rightarrow a_i a_{i+1} \ldots a_{i+j-1}$, that is, A derives the substring of w beginning at position i and of length j. To determine whether the string w is in $L(G)$, the algorithm computes the parse table T and look to see whether S is in entry $t_{1,n}$.

In the first step of constructing the parse table, the CYK algorithm sets $t_{i,1} = \{ A | A \rightarrow a_i$ is in $P\}$. In the jth step, the algorithm assumes that $t_{i,j'}$ has been computed for $1 \leq i \leq n$ and $1 \leq j' < j$, and it computes $t_{i,j}$ by examining the nonterminals in the following pairs of entries:

$$(t_{i,1}, t_{i+1,j-1}), (t_{i,2}, t_{i+2,j-2}), \ldots,$$
$$(t_{i,j-1}, t_{i+j-1,1}),$$

5	S, A				
4	S, A	S, A			
3	S, A	S	S, A		
2	S	A	S	S	
$j = 1$	A	S	A	A	A
	$i = 1$	2	3	4	5
	a	b	a	a	a

Probabilistic Context-Free Grammars, Fig. 2 The parse table T of G for "*abaaa*"

and if B is in $t_{i,k}$ and C is in $t_{i+k,j-k}$ for some k $(1 \leq k < j)$ and the production $A \rightarrow BC$ is in P, A is added to $t_{i,j}$.

For example, we consider a simple CFG $G = (N, \Sigma, P, S)$ of Chomsky normal form where $N = \{S, A\}$, $\Sigma = \{a, b\}$ and

$$P = \{S \rightarrow AA, S \rightarrow AS, S \rightarrow b,$$
$$A \rightarrow SA, A \rightarrow a\}.$$

This CFG generates a string "*abaaa*," that is, $S \Rightarrow abaaa$, and the parse table T for *abaaa* is shown in Fig. 2. The parse table can efficiently store all possible parse trees of G for *abaaa*.

Learning

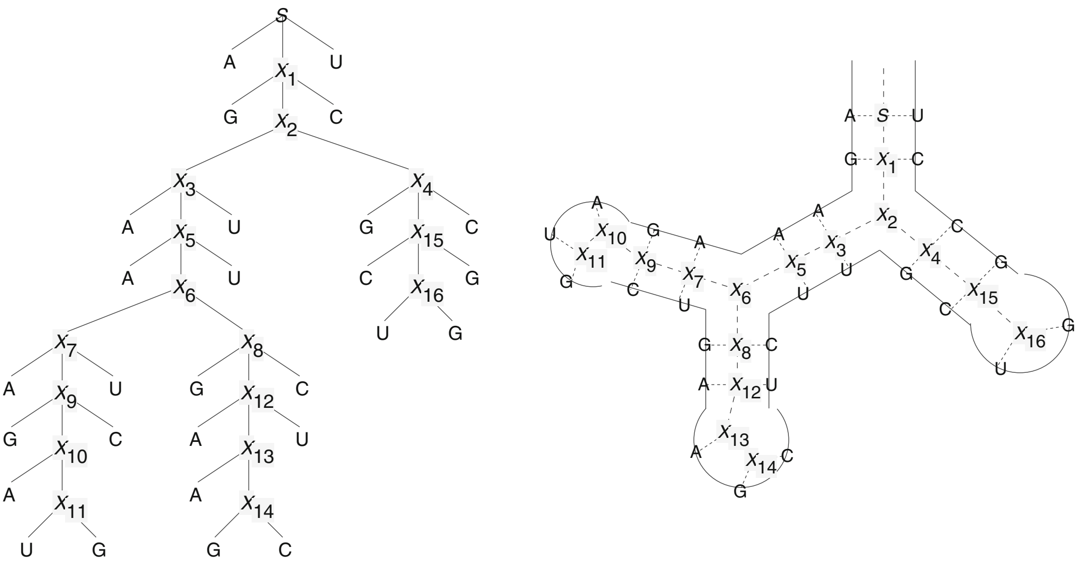
The problem of learning PCFGs from example strings has two aspects: determining a discrete structure (topology) of the target grammar and estimating probabilistic parameters in the grammar (Sakakibara 1997). Based on the maximum likelihood criterion, an efficient estimation algorithm for probabilistic parameters has been proposed: the inside-outside algorithm for PCFGs. On the other hand, finding an appropriate discrete structure of a grammar is a harder problem.

The procedure to estimate the probabilistic parameters of a PCFG is known as the *inside-outside* algorithm. Just like the forward-backward algorithm for HMMs, this procedure is an expectation-maximization (EM) method for obtaining maximum likelihood of the grammar's parameters. However, it requires the grammar to be in Chomsky normal form, which is inconvenient to handle in many practical problems (and requires more nonterminals). Further, it takes time at least proportional to n^3, whereas the forward-backward procedure for HMMs takes time proportional to n^2, where n is the length of the string w. There are also many local maxima in which the method can get caught. Therefore, the initialization of the iterative process is crucial since it affects the speed of convergence and the goodness of the results.

Application to Bioinformatics

An effective method for learning and building PCFGs has been applied to modeling a family of RNA sequences (Durbin et al. 1998; Sakakibara 2005). In RNA, the nucleotides adenine (A), cytosine (C), guanine (G), and uracil (U) interact in specific ways to form characteristic secondary-structure motifs such as helices, loops, and bulges. In general, the folding of an RNA chain into a functional molecule is largely governed by the formation of intramolecular A-U and G-C Watson–Crick pairs. Such base pairs constitute the so-called biological palindromes in a genome and can be clearly described by a CFG. In particular, productions of the forms $X \rightarrow$ A Y U, $X \rightarrow$ U Y A, $X \rightarrow$ G Y C, and $X \rightarrow$ C Y G describe a structure in RNA due to Watson–Crick base pairing. Using productions of this type, a CFG can specify a language of biological palindromes.

For example, the application of productions in the grammar shown in Fig. 1 generates the RNA sequence CAUCAGGGAAGAUCUCUUG and the derivation can be arranged in a tree structure of a *parse tree* (Fig. 3, left). A parse tree represents the syntactic structure of a sequence produced by a grammar. For the RNA sequence, this syntactic structure corresponds to the physical secondary structure (Fig. 3, right). PCFGs are applied to perform three tasks in RNA sequence analysis: to discriminate RNA-family sequences from nonfamily sequences, to produce multiple alignments, and to ascertain the secondary structure of new sequences.



Probabilistic Context-Free Grammars, Fig. 3 A parse tree (*left*) generated by a simple context-free grammar (CFG) for RNA molecules and the physical secondary structure (*right*) of the RNA sequence which is a reflection of the parse tree

Recommended Reading

Durbin R, Eddy S, Krogh A, Mitchison G (1998) Biological sequence analysis. Cambridge University Press, Cambridge

Hopcroft JE, Ullman JD (1979) Introduction to automata theory, languages and computation. Addison-Wesley, Reading

Lari K, Young SJ (1990) The estimation of stochastic context-free grammars using the inside-outside algorithm. Comput Speech Lang 4:35–56

Sakakibara Y (1997) Recent advances of grammatical inference. Theor Comput Sci 185:15–45

Sakakibara Y (2005) Grammatical inference in bioinformatics. IEEE Trans Pattern Anal Mach Intell 27:1051–1062

Probability Calibration

▶ Classifier Calibration

Probably Approximately Correct Learning

▶ PAC Learning

Process-Based Modeling

▶ Inductive Process Modeling

Program Synthesis from Examples

▶ Inductive Programming

Programming by Demonstration

Pierre Flener[1] and Ute Schmid[2]
[1]Department of Information Technology,
Uppsala University, Uppsala, Sweden
[2]Faculty of Information Systems and Applied
Computer Science, University of Bamberg,
Bamberg, Germany

Abstract

Programming by demonstration (PBD) is introduced as family of approaches to teach a computer system new behavior by demonstrating it in the context of a concrete example.

Most of the work by this author was done while on leave of absence in 2006/2007 as a Visiting Faculty Member and Erasmus Exchange Teacher at Sabancı University, Turkey.

References to classical and current PBD systems are given.

Synonyms

Programming by example (PBE)

Definition

Programming by demonstration (PBD) describes a collection of approaches for the support of end-user programming with the goal of making the power of computers fully accessible to all users. The general objective is to *teach* computer systems new behavior by demonstrating (repetitive) actions on concrete examples. A user provides examples of how a program should operate, either by demonstrating trace steps or by showing examples of the inputs and outputs, and the system infers a generalized program that achieves those examples and can be applied to new examples. Typical areas of application are macro generation (e.g., for text editing), simple arithmetic functions in spreadsheets, simple shell programs, XML transformations, or query-replace commands, as well as the generation of helper programs for web agents, geographic information systems, or computer-aided design. The most challenging approach to PBD is to obtain generalizable examples by minimal intrusion, where the user's ongoing actions are recorded without an explicit signal for the start of an example and without explicit confirmation or rejection of hypotheses. An early example of such a system is EAGER (Cypher 1993a).

Current PBD approaches incorporate some simple forms of **generalization** learning, but typically no or only highly problem-dependent methods for the induction of loops or recursion from examples or traces of repetitive commands. Introducing **inductive programming** or **trace-based programming** methods into PBD applications could significantly increase the possibilities of end-user programming support. This is demonstrated impressively with the Microsoft Excel plug-in Flash Fill (Gulwani et al. 2012).

Cross-References

- ▶ Inductive Programming
- ▶ Trace-Based Programming

Recommended Reading

Cypher A (1993a) Programming repetitive tasks by demonstration. In: Cypher A (ed) Watch what I do: programming by demonstration. MIT, Cambridge, pp 205–217

Cypher A (ed) (1993b) Watch what I do: programming by demonstration. MIT, Cambridge

Gulwani S, Harris WR, Singh R (2012) Spreadsheet data manipulation using examples. Commun ACM 55(8):97–105

Lieberman H (ed) (2001) Your wish is my command: programming by example. Morgan Kaufmann, San Francisco

Programming by Example (PBE)

▶ Programming by Demonstration

Programming by Examples

▶ Inductive Programming

Programming from Traces

▶ Trace-Based Programming

Projective Clustering

Cecilia M. Procopiuc
AT&T Labs, NJ, USA

Synonyms

Local feature selection; Subspace clustering

Definition

Projective clustering is a class of problems in which the input consists of high-dimensional data, and the goal is to discover those subsets of the input that are strongly correlated in subspaces of the original space. Each subset of correlated points, together with its associated subspace, defines a *projective cluster*. Thus, although all cluster points are close to each other when projected on the associated subspace, they may be spread out in the full-dimensional space. This makes projective clustering algorithms particularly useful when mining or indexing datasets for which full-dimensional clustering is inadequate (as is the case for most high-dimensional inputs). Moreover, such algorithms compute projective clusters that exist in different subspaces, making them more general than global dimensionality-reduction techniques.

Motivation and Background

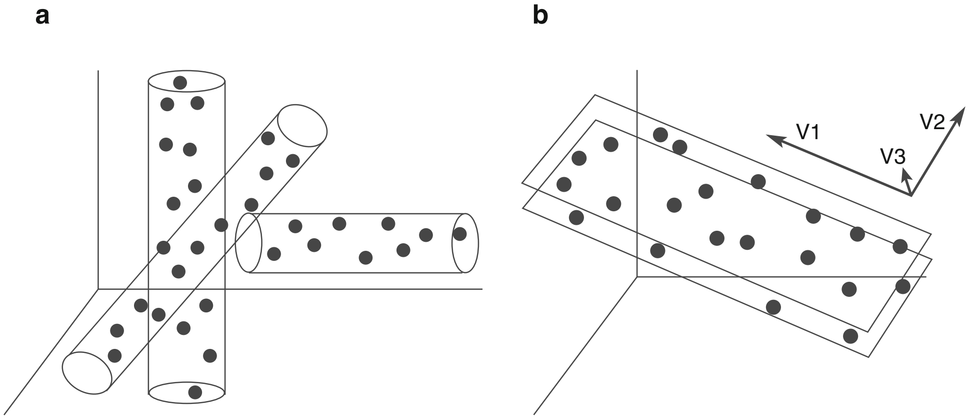
Projective clustering is a type of data mining whose main motivation is to discover correlations in the input data that exist in subspaces of the original space. This is an extension of traditional full-dimensional clustering, in which one tries to discover point subsets that are strongly correlated in all dimensions. Figure 1a shows an example of input data for which full-dimensional clustering cannot discover the three underlying patterns. Each pattern is a projective cluster.

It is well known (Beyer et al. 1999) that for a broad class of data distributions, as the dimensionality increases, the distance to the nearest neighbor of a point approaches the distance to its farthest neighbor. This implies that full-dimensional clustering will fail to discover significantly correlated subsets on such data, since the diameter of a cluster is almost the same as the diameter of the entire dataset. In practice, many applications from text and image processing generate data with hundreds or thousands of dimensions, which makes them extremely bad candidates for full-dimensional clustering methods.

One popular technique to classify high-dimensional data is to first project it onto a much lower-dimensional subspace, and then employ a full-dimensional clustering algorithm in that space. The projection subspace is the same for all points, and is computed so that it best "fits" the data. A widely used dimensionality-reduction technique, called ▸ principal component analysis (PCA), defines the best projection subspace to be the one that minimizes least-square error. While this approach has been proven successful in certain areas such as text mining, its effectiveness depends largely on the characteristics of the data. The reason is that there may be no way to choose a single projection subspace without encountering a significant error; or alternatively, setting a maximum bound on the error results in a subspace with high dimensionality. Figure 1b shows the result of PCA on a good candidate set. The points are projected on the subspace spanned by vectors V_1 and V_2, along which they have greatest variance. However, for the example in Fig. 1a, no plane or line fits the data well enough. Projective clustering can thus be viewed as a generalized dimensionality-reduction method, in which different subsets of the data are projected on different subspaces.

There are many variants of projective clustering, depending on what quality measure one tries to optimize for the clustering. Most such measures, however, are expressed as a function of the distances between points in the clusters. The distance between two cluster points is computed with respect to the subspace associated with that cluster. Alternative quality measures consider the density of cluster points inside the associated subspace.

Megiddo and Tamir (1982) showed that it is NP-Hard to decide whether a set of n points in the plane can be covered by k lines. This early result implies not only that most projective clustering problems are NP-Complete even in the planar case, but also that approximating the objective function within a constant factor is NP-Complete. Nevertheless, several approximation algorithms have been proposed, with running time polyno-



Projective Clustering, Fig. 1 Dimensionality reduction via (**a**) projective clustering and (**b**) principal component analysis

mial in the number of points n and exponential in the number of clusters k. Agrawal et al. (1998) proposed a subspace clustering method based on density measure that computes clusters in a bottom-up approach (from lower to higher dimensions). Aggarwal et al. (1999) designed a partitioning-style algorithm.

Theory

Many variants of projective clustering problems use a distance-based objective function and thus have a natural geometric interpretation. In general, the optimization problem is stated with respect to one or more parameters that constrain the kind of projective clusters one needs to investigate. Examples of such parameters are: the number of clusters, the dimensionality (or average dimensionality) of the clusters, the maximum size of the cluster in its associated subspace, the minimum density of cluster points, etc. Below we present the most frequently studied variants for this problem.

Distance-Based Projective Clustering

Given a set S of n points in $\mathbb{R}^d$ and two integers $k < n$ and $q \leq d$, find kq-dimensional flats $h_1, \ldots, h_k$ and partition S into k subsets $C_1, \ldots, C_k$ so that one of the following objective functions is minimized:

$$\max_{1 \leq i \leq k} \max_{p \in C_i} d(p, h_i) \qquad (k-\text{center})$$

$$\sum_{1 \leq i \leq k} \sum_{p \in C_i} d(p, h_i) \qquad (k-\text{median})$$

$$\sum_{1 \leq i \leq k} \sum_{p \in C_i} d(p, h_i) \qquad (k-\text{means})$$

These types of problems are also referred to as *geometric clustering problems*. They require all cluster subspaces to have the same dimensionality, i.e., $d - q$ (the subspace associated with C_i is orthogonal to h_i). The number of clusters is also fixed, and the clustering must be a partitioning of the original points.

Further variants are defined by introducing slight modifications in the above framework. For example, one can allow the existence of outliers, i.e., points that do not belong to any projective cluster. This is generally done by providing an additional parameter, which is the maximum percentage of outliers. The problems can also be changed to a dual formulation, in which a maximum value for the objective function is specified, and the goal is to minimize the number of clusters k.

Special cases for the k-center objective function are $q = d - 1$ and $q = 1$. In the first case, the problem is equivalent to finding k hyper-strips that contain S so that the maximum width of a hyper-strip is minimized. If $q = 1$, then the problem is to cover S by k congruent hyper-cylinders of smallest radius. Since this is equivalent to finding the k lines that are the axes of the hyper-cylinders, this problem is also referred to as *k-line-center*. Figure 1a is an example of 3-line-center.

In addition, k-median problems have also been studied when cluster subspaces have different dimensionalities. In that case, distances computed in each cluster are normalized by the dimensionality of the corresponding subspace.

Density-Based Projective Clustering

A convex region in a subspace is called dense if the number of data points that project inside it is larger than some user-defined threshold. For a fixed subspace, the convex regions of interest in that subspace are defined in one of several ways, as detailed below. Projective clusters are then defined to be connected unions of dense regions of interest. The different variants for defining regions of interest can be broadly classified in three classes:

(*ε-Neighborhoods*) Regions of interest are L_p-balls of radius ε centered at the data points. In general, L_p is either L_2 (hyper-spheres) or L_∞ (hyper-cubes).

(Regular Grid Cells) Regions of interest are cells defined by an axis-parallel grid in the subspace. The grid hyper-planes are equidistant along each dimension.

(Irregular Grid Cells) Regions of interest are cells defined by an irregular grid in the subspace. Parallel grid hyper-planes are not necessarily equidistant, and they may also be arbitrarily oriented.

Another variant of projective clustering defines a so-called *quality measure* for a projective cluster, which depends both on the number of cluster points and the number of dimensions in the associated subspace. The goal is to compute the clusters that maximize this measure. Projective clusters are required to be L_p-balls of fixed radius in their associated subspace, which means that clusters in higher dimensions tend to have fewer points, and vice-versa. Hence, the quality measure provides a way to compare clusters that exist in different number of dimensions. It is related to the notion of dense ε-neighborhoods.

Many other projective clustering problems are application driven and do not easily fit in the above classification. While they follow the general framework of finding correlations among data in subspaces of the original space, the notion of projective cluster is specific to the application. One such example is presented later in this section.

Algorithms

Distance-based projective clustering problems are NP-Complete when the number of clusters k is an input parameter. Moreover, k-center problems cannot be approximated within a constant factor, unless $P = NP$. This follows from the result of Meggido and Tamir (1982), who showed that it is NP-Hard to decide whether a set of n points in the plane can be covered by k lines.

Agarwal and Procopiuc (2003) first proposed approximation algorithms for k-center projective clustering in two and three dimensions. The algorithms achieve constant factor approximation by generating more clusters than required.

Subsequent work by several other authors led to the development of a general framework in which $(1 + \varepsilon)$-approximate solutions can be designed for several types of full-dimensional and projective clustering. In particular, k-center and k-means projective clustering can be approximated in any number of dimensions. The idea is to compute a so-called *coreset*, which is a small subset of the points, such that the optimal projective clusters for the coreset closely approximate the projective clusters for the original set. Computing the optimal solution for the coreset has (super) exponential dependence on the number of clusters k, but it is significantly faster than computing the optimal solution for the original set of points. The survey by Agarwal et al. (2005) gives a comprehensive overview of these results.

While the above algorithms have approximation guarantees, they are not practical even for moderate values of n, k, and d. As a result, heuristic methods have also been developed for these problems. The general approach is to iteratively refine a current set of clusters, either by re-assigning points among them, or by merging nearby clusters. When the set of points in a cluster changes, the new subspace associated with the cluster is also recomputed, in a way that tries to optimize the objective function for the new clustering. Aggarwal et al. (1999) proposed the PROCLUS algorithm for k-median projective clustering with outliers. The cluster subspaces can have different dimensionalities, but they must be orthogonal to coordinate axes. Aggarwal and Yu (2000) subsequently extended the algorithm to arbitrarily oriented clusters, but with the same number of dimensions. Agarwal and Mustafa (2004) proposed a heuristic approach for k-means projective clustering with arbitrary orientation and different dimensionalities.

The first widely used method for density-based projective clustering was proposed by Agrawal et al. (1998). The algorithm, called CLIQUE, computes projective clusters based on regular grid cells in orthogonal subspaces, starting from the lowest-dimensional subspaces (i.e., the coordinate axes) and iterating to higher dimensions. Pruning techniques are used to skip subspaces in which a large fraction of points lie outside dense regions. Subsequent strategies improved the running time and accuracy by imposing irregular grids and using different pruning criteria.

Böhm et al. (2004) designed an algorithm called 4C for computing density-connected ε-neighborhoods in arbitrarily oriented subspaces. The method is agglomerative: It computes the local dimensionality around each point p by using PCA on all points inside the (full-dimensional) ε-neighborhood of p. If the dimensionality is small enough and the neighborhood is dense, then p and its neighbors form a projective cluster. Connected projective clusters with similarly oriented subspaces are then repeatedly merged.

The OptiGrid algorithm by Hinneburg and Keim (1999) was the first method to propose irregular grid cells of arbitrary (but fixed) orientation. Along each grid direction, grid hyper-planes are defined to pass through the local minima of a probability density function. This significantly reduces the number of cells compared with a regular grid that achieves similar overall accuracy. The probability density function is defined using the kernel-density estimation framework. Input points are projected on the grid direction, and their distribution is extrapolated to the entire line by the density function

$$f(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - s_i}{h}\right),$$

where $s_1, \ldots, s_n$ denote the projections of the input points, and h is a parameter. The function $K(x)$, called the *kernel*, is usually the Gaussian function, although other kernels can also be used.

The DOC algorithm proposed by Procopiuc et al. (2002) approximates optimal clusters for a class of quality measures. Orthogonal projective clusters are computed iteratively via random sampling. If a sample is fully contained in a cluster then it can be used to determine the subspace of that cluster, as well as (a superset of) the other cluster points. Such a sample is called a *discriminating set*. Using the properties of the quality measure, the authors show that a discriminating set is found with high probability after a polynomial number of trials.

An overview of most of these practical methods, as well as of subsequent work expanding their results, can be found in the survey by Parsons et al. (2004).

Applications

Similar to full-dimensional clustering, projective clustering methods provide a way to efficiently organize databases for searching, as well as for pattern discovery and data compression. In a broad sense, they can be used in any application that handles high-dimensional data, and which can benefit from indexing or mining capabilities. In practice, additional domain-specific information is often necessary. We present an overview of the generic database usage first, and then discuss several domain-specific applications.

Data Indexing

An index tree is a hierarchical structure defined on top of a data set as follows. The root corresponds to the entire data set. For each internal node, the data corresponding to that node is partitioned in some pre-defined manner, and there is a child of the node corresponding to each subset in the partition. Often, the partitioning method is a distance-based clustering algorithm. In addition, each node stores the boundary of a geometric region that contains its points, to make searching the structure more efficient. For many popular indexes, the geometric region is the minimum axis-parallel bounding box. Index trees built with full-dimensional clustering methods become inefficient for dimensionality about 10 or higher, due to the large overlap in the geometric regions of sibling nodes. Chakrabarti and Mehrotra (2000) first proposed an index tree that uses projective clustering as a partitioning method. In that case, each node also stores the subspace associated with the cluster.

Pattern Discovery

A projective cluster, by definition, is a pattern in the data, so any of the above algorithms can be used in a pattern discovery application. However, most applications restrict the projective clusters to be orthogonal to coordinate axes, since the axes have special interpretations. For example, in a database of employees, one axis may represent salary, another the length of employment, and the third one the employees' age. A projective cluster in the subspace spanned by salary and employment length has the following interpretation: there is a correlation between salaries in range A and years of employment in range B, which is independent of employees' age.

Data Compression

As discussed in the introduction, projective clusters can be used as a dimensionality-reduction technique, by replacing each point with its projection on a lower dimensional subspace. The projection subspace is orthogonal to the subspace of the cluster that contains the point. In general, this method achieves smaller information loss and higher compression ratio than a global technique such as PCA.

Image Processing

A picture can be represented as a high-dimensional data point, where each pixel represents one dimension, and its value is equal to the RGB color value of the pixel. Since this representation loses pixel adjacency information, it is generally used in connection with a smoothing technique, which replaces the value of a pixel with a function that depends both on the old pixel value, and the values of its neighbors. A projective cluster groups images that share some similar features, while they differ significantly on others. The DOC algorithm has been applied to the face detection problem as follows: Projective clusters were computed on a set of (pre-labeled) human faces, then used in a classifier to determine whether a new image contained a human face.

Document Processing

Text documents are often represented as sparse high-dimensional vectors, with each dimension corresponding to a distinct word in the document collection. Several methods are used to reduce the dimensionality, e.g., by eliminating so-called stop words such as "and," "the," and "of." A non zero entry in a vector is usually a function of the corresponding word's frequency in the document. Because of the inherent sparsity of the vectors, density-based clustering, as well as k-center methods, are poor choices for such data. However, k-means projective clustering has been successfully applied to several document corpora (Li et al. 2004).

DNA Microarray Analysis

A gene-condition expression matrix, generated by a DNA microarray, is a real-valued matrix, such that each row corresponds to a gene, and each column corresponds to a different condition. An entry in a row is a function of the relative abundance of the mRNA of the gene under that specific condition. An orthogonal projective cluster thus represents several genes that have similar expression levels under a subset of conditions. Genetics researchers can infer connections between a disease and the genes in a cluster. Due to the particularities of the data, different notions of similarity are often required. For example, order preserving clusters group genes that have the same tendency on a subset of attributes, i.e., an attribute has the same rank (rather than similar value) in each projected gene. See the results of Liu and Wang (2003).

Principal Component Analysis

PCA also referred to as the Karhunen-Loève Transform, is a global ▶ dimensionality reduction technique, as opposed to projective clustering, which is a local dimensionality reduction method. PCA is defined as an orthogonal linear transformation with the property that it transforms the data into a new coordinate system, such that the projection of the data on the first coordinate has the greatest variance among all projections on a line, the projection of the data on the second coordinate has the second greatest variance, and so on. Let X denote the data matrix, with each point written as a column vector in X, and modified so that X has empirical mean zero (i.e., the mean vector is subtracted from each data point). Then the eigenvectors of the matrix XX T are the coordinates of the new system. To reduce the dimensionality, keep only the eigenvectors corresponding to the largest few eigenvalues.

Coresets

Let $P \subseteq \mathbb{R}^d$ be a set of points, and μ be a measure function defined on subsets of $\mathbb{R}^d$, such that μ is monotone (i.e., for $P_1 \subseteq P_2$, $\mu(P_1) \leq \mu(P_2)$). A subset $Q \subseteq P$ is an *ε-coreset* with respect to μ if $(1-\varepsilon)\mu(P) \leq \mu(Q)$. The objective functions for k-center, k-median, and k-means projective clustering are all examples of measure functions μ.

Cross-References

▶ Clustering
▶ Curse of Dimensionality
▶ Dimensionality Reduction
▶ Kernel Methods
▶ K-Means Clustering
▶ Principal Component Analysis

Recommended Reading

Agarwal PK, Mustafa N (2004) k-means projective clustering. In: Proceeding of ACM SIGMOD-SIGACT-SIGART symposium principles of database systems, pp 155–165

Agarwal PK, Procopiuc CM (2003) Approximation algorithms for projective clustering. J Algorithms 46(2):115–139

Agarwal PK, Har-Peled S, Varadarajan KR (2005) Geometric approximation via coresets. In: Goodman JE, Pach J, Welzl E (eds) Combinatorial and computational geometry. Cambridge University Press, Cambridge/New York, pp 1–30

Aggarwal CC, Yu PS (2000) Finding generalized projected clusters in high dimensional spaces. In: Proceeding of ACM SIGMOD international conference management of data, pp 70–81

Aggarwal CC, Procopiuc CM, Wolf JL, Yu PS, Park JS (1999) Fast algorithms for projected clustering. In: Proceeding of ACM SIGMOD international conference management of data, pp 61–72

Agrawal R, Gehrke J, Gunopulos D, Raghavan P (1998) Automatic subspace clustering of high dimensional data for data mining applications. In: Proceeding of ACM SIGMOD international conference management of data, pp 94–105

Beyer K, Goldstein J, Ramakrishnan R, Shaft U (1999) When is "nearest neighbour" meaningful? In: Proceeding of 7th international conference data theory, vol 1540, pp 217–235

Böhm C, Kailing K, Kröger P, Zimek A (2004) Computing clusters of correlation connected objects. In: Proceeding of ACM SIGMOD international conference management of data, pp 455–466

Chakrabarti K, Mehrotra S (2000) Local dimensionality reduction: a new approach to indexing high dimensional spaces. In: Proceeding of 26th international conference very large data bases, pp 89–100

Hinneburg A, Keim DA (1999) Optimal grid-clustering: towards breaking the curse of dimensionality in high-dimensional clustering. In: Proceeding of 25th international conference very large data bases, pp 506–517

Li T, Ma S, Ogihara M (2004) Document clustering via adaptive subspace iteration. In: Proceeding of 27th international ACM SIGIR conference research and development in information retrieval, pp 218–225

Liu J, Wang W (2003) Op-cluster: clustering by tendency in high dimensional space. In: Proceeding of international conference on data mining, pp 187–194

Megiddo N, Tamir A (1982) On the complexity of locating linear facilities in the plane. Oper Res Lett 1:194–197

Parsons L, Haque E, Liu H (2004) Subspace clustering for high dimensional data: a review. ACM SIGKDD Explor Newslett 6(1):90–105

Procopiuc CM, Jones M, Agarwal PK, Murali TM (2002) A Monte Carlo algorithm for fast projective clustering. In: Proceeding of ACM SIGMOD international conference management of data, pp 418–427

Prolog

Prolog is a declarative programming language based on logic. It was conceived by French and British computer scientists in the early 1970s. A considerable number of public-domain and commercial Prolog interpreters are available today. Prolog is particularly suited for applications requiring pattern matching or search. Prolog programs are also referred to as ▶ logic programs.

In machine learning, classification rules for structured individuals can be expressed using a subset of Prolog. Learning Prolog programs from examples is called ▶ inductive logic programming (ILP). ILP systems are sometimes – but not always – implemented in Prolog. This has the advantage that classification rules can be executed directly by the Prolog interpreter.

Cross-References

- ▶ Clause
- ▶ First-Order Logic
- ▶ Inductive Logic Programming
- ▶ Logic Program

Recommended Reading

Colmerauer A, Kanoui H, Pasero R, Roussel P (1973) Un système de communication homme-machine an Français. Report, Groupè d'Intelligence Artificielle, University d'Aix Marseille II, Luminy

Kowalski RA (1972) The predicate calculus as a programming language. In: Proceedings of the international symposium and summer school on mathematical foundations of computer science, Jablonna

Roussel P (1975) Prolog: Manual de reference et d'utilization. Technical report, Groupe d'Intelligence Artificielle, Marseille-Luminy

Property

▶ Attribute

Propositional Logic

Propositional logic is the logic of propositions, i.e., expressions that are either true or false. Complex propositions are built from propositional atoms using logical connectives. Propositional logic is a special case of predicate logic, where all ▶ predicates have zero arity; see the entry on first-order logic for details.

Cross-References

- ▶ First-Order Logic
- ▶ Propositionalization

Propositionalization

Nicolas Lachiche
University of Strasbourg, Strasbourg, France

Abstract

Propositionalization is the process of explicitly transforming a ▶ relational dataset into a propositional dataset.

Definition

The input data consists of examples represented by structured terms (cf. ▶ learning from structured data), several predicates in ▶ first-order logic, or several tables in a relational database. We will jointly refer to these as *relational representations*. The output is an ▶ attribute-value representation in a single table, where each example corresponds to one row and is described by its values for a fixed set of attributes. New attributes are often called features to emphasize that they are built from the original attributes. The aim of propositionalization is to preprocess relational data for subsequent analysis by attribute-value learners. There are several reasons for doing this, the most important of which are to reduce the complexity and speed up the learning, to separate modeling the data from hypothesis construction, or to use familiar attribute-value (or propositional) learners.

Motivation and Background

Most domains are naturally modeled by several tables in a relational database or several classes in an object-oriented language, for example, customers and their transactions; molecules, their atoms, and bonds; or patients and their examinations. A proper relational dataset involves at least two tables linked together. Typically, one table of the relational representation corresponds to the individuals of interest for the machine learning task, and the other tables contain related information that could be useful. The first table is the individual or the primary table; the other tables are complementary tables.

Example 1 Let us consider a simplified medical domain as an example. This is inspired by a real medical dataset (Tomečková et al. 2002). It consists of four tables.

The patient table is the primary table. It contains data on each patient such as the patient identifier (pid), name, date of birth, height, job, the identifier of the company where the patient works, etc.

Patient

pid	Name	Birth	Height	Job	Company	…
I	Smith	15/06/1956	1.67	Manager	a	…
II	Blake	13/02/1968	1.82	Salesman	a	…
⋮	⋮	⋮	⋮	⋮	⋮	…

The company table contains its name, its location, and so on. There is a many-to-one relationship from the patient table to the company table: A patient works for a single company, but a company may have several employees.

Company

cid	Name	Location	…
a	Eiffel	Paris	…
⋮	⋮	⋮	…

The examination table contains the information on all examinations of all patients. For each examination, its identifier (eid), the patient identifier (pid), the date, the patient's weight, whether the patient smokes, his or her blood pressure, etc. are recorded. Of course, each examination corresponds to a single patient, and a given patient can have several examinations, i.e., there is a one-to-many relationship from the patient table to the examination table.

Examination

eid	pid	Date	Weight	Smokes	BP	…
1	I	10/10/1991	60	Yes	10	…
2	I	04/06/1992	64	Yes	12	…
⋮	⋮	⋮	⋮	⋮	⋮	…
23	II	20/12/1992	80	Yes	10	…
24	II	15/11/1993	78	No	11	…
⋮	⋮	⋮	⋮	⋮	⋮	…

Additional tests can be prescribed at each examination. Their identifiers (tid), corresponding examinations (eid), names, values, and interpretations are recorded in the additional_test table.

Additional_test

tid	eid	Date	Name	Value	Interpretation
t237	1	19/10/1991	Red blood cells	35	Bad
t238	1	23/10/1991	Radiography	Nothing	Good
⋮	⋮	⋮	⋮	⋮	⋮
t574	2	07/06/1992	Red blood cells	43	Good
⋮	⋮	⋮	⋮	⋮	⋮

Several approaches exist to deal directly with relational data, e.g., ▶ inductive logic programming, ▶ relational data mining (Džeroski and Lavrač 2001), or ▶ statistical relational learning. However relational hypotheses can be transformed into propositional expressions.

Generally, a richer representation language permits the description of more complex concepts; however, the cost of this representational power is that the search space for learning greatly increases. Therefore, mapping a relational representation into a propositional one generally reduces search complexity.

A second motivation of propositionalization is to focus on the construction of features before combining them into a hypothesis (Srinivasan et al. 1996). This is related to ▶ feature construction and to the use of background knowledge. One could say that propositionalization aims at building an intermediate representation of the data in order to simplify the hypothesis subsequently found by a propositional learner.

A third motivation is pragmatic. Most available machine learning systems deal with propositional data only, but tend to include a range of algorithms in a single environment, whereas relational learning systems tend to concentrate on a single algorithm. Propositional systems are therefore often more versatile and give users the possibility to work with the algorithms they are used to.

Solutions

There are various ways to propositionalize relational data consisting of at least two tables linked together through a relationship. We will first focus on a single relationship between two tables. Most approaches can then iteratively deal with several relationships as explained below.

Propositionalization mechanisms depend on whether that relationship is functional or nondeterminate. This distinction explains most common mistakes made by newcomers.

Functional Relationship (Many-to-One, One-to-One)

When the primary table has a many-to-one or one-to-one relationship to the complementary table, each row of the primary table links to one row of the complementary table. A simple join of the two tables results in a single table where each row of the primary table is completed with the information derived from the complementary table.

Example 2 In our simplified medical domain, there is a many-to-one relationship from each patient to his or her company. Let us focus on those two tables only. A join of the two tables results in a single table where each row describes a single patient and the company he or she works for.

Patient and his/her company

pid	Name	Birth	Height	Job	cid	Company	Location	...
I	Smith	15/06/1956	1.67	Manager	a	Eiffel	Paris	...
II	Blake	13/02/1968	1.82	Salesman	a	Eiffel	Paris	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	...

The resulting table is suitable for any attribute-value learner

Nondeterminate Relationship (One-to-Many, Many-to-Many)

Propositionalization is less trivial in a nondeterminate context, when there is a one-to-many or many-to-many relationship from the primary table to the complementary table, i.e., when one individual of the primary table is associated with a set of rows of the complementary table.

A propositional attribute is built by applying an aggregation function to a column of the complementary table over a selection of rows. Of course a lot of conditions can be used to select the rows. Those conditions can involve other columns than the aggregated column. Any aggregation function can be used, e.g., to check whether the set is not empty, to count how many elements there are, to find the mean (for numerical) or the mode (for categorical) values, etc.

Example 3 In our simplified medical domain, there is a one-to-many relationship from the patient to his or her examinations. Let us focus on those two tables only. Many features can be constructed. Simple features are aggregation functions applied to a scalar (numerical or categorical) column. The number of occurrences of the different values of every categorical attributes can be counted. For instance, the f60 feature in the table below counts in how many examinations the patient stated he or she smoked. The maximum, minimum, average, and standard deviation of every numerical column can be estimated, e.g., the f84 and f85 features in the table below, respectively, estimate the average and the maximum blood pressure of the patient over his or her examinations. The aggregation functions can be applied to any selection of rows, e.g., the f135 feature in the table below estimates the average blood pressure over the examinations when the patient smoked.

Patient and his/her examinations

pid	Name	…	f60	…	f84	f85	…	f135	…
I	Smith	…	2	…	11	12	…	11	…
II	Blake	…	1	…	10.5	11	…	10	…
⋮	⋮	…	⋮	…	⋮	⋮	…	⋮	…

From this example it is clear that nondeterminate relationships can easily lead to a combinatorial explosion of the number of features.

Common Mistakes and Key Rules to Avoid Them

Two mistakes are frequent when machine learning practitioners face a propositionalization problem, i.e., when they want to apply a propositional learner to an existing relational dataset (Lachiche 2005).

The first mistake is to misuse the (universal) join. Join is valid in a functional context, as explained earlier. When applied to a nondeterminate relationship, it produces a table where several rows correspond to a single individual, leading to a multiple-instance problem (Dietterich et al. 1997) (cf. ▶ multi-instance learning).

Example 4 In our simplified medical domain, there is a one-to-many relationship from the patient table to the examination table. If a join is performed, each row of the examination table is completed with the information on the examined patient, i.e., there are as many rows as examinations.

Examination and its patient

eid	Date	Weight	Smokes	BP	…	pid	Name	…
1	10/10/1991	60	Yes	10	…	I	Smith	…
2	04/06/1992	64	Yes	12	…	I	Smith	…
⋮	⋮	⋮	⋮	⋮	…	⋮	⋮	…
23	20/12/1992	80	Yes	10	…	II	Blake	…
24	15/11/1993	78	No	11	…	II	Blake	…
⋮	⋮	⋮	⋮	⋮	…	⋮	⋮	…

In this example, the joined table deals with the examinations rather than with the patients. An attribute-value learner could be used to learn hypotheses about the examinations, not about the patients

This example reinforces a key representation rule in attribute-value learning: "Each row corresponds to a single individual, and vice-versa."

The second mistake is a meaningless column concatenation. This is more likely when a one-to-many relationship can be misinterpreted as several one-to-one relationships, i.e., when the practitioner is led to think that a nondeterminate relationship is actually functional.

Example 5 In our simplified medical domain, let us assume that the physician numbered the successive examinations (1, 2, 3, and so on) of each patient. Then given that each patient has a first examination, it is tempting to consider that there is a functional relationship from the patient to his or her "first" examination, "second" examination, and so on. This would result in a new patient table with concatenated columns: weight at the first examination, whether he or she smoked at the first examination, . . . , weight at the second examination, etc. This could easily lead

Patient and his/her examinations (incorrect representation!)

pid	Name	…	"First" examination			"Second" examination			…
			Weight	Smokes	…	Weight	Smokes	…	…
I	Smith	…	60	Yes	…	64	Yes	…	…
II	Blake	…	80	Yes	…	78	No	…	…
⋮	⋮	…	⋮	…	⋮	⋮	⋮	…	…

to an attribute-value learner generalizing over a patient's weight at their *i*th examination, which is very unlikely to be meaningful

Two aspects should warn the user of such a representation problem: first, the number of columns depends on the dataset, and as a consequence, lots of columns are not defined for all individuals. Moreover, when the absolute numbering does not make sense, there is no functional relationship. Such a misunderstanding can be avoided by remembering that in an attribute-value representation, "each column is uniquely defined for each row."

Further Relationships

The first complementary table can itself have a nondeterminate relationship with another complementary table and so on. Two approaches are available.

A first approach is to consider the first complementary table, the one having a one-to-many relationship, as a new primary table in a recursive propositionalization.

Example 6 In our simplified medical domain, the examination table has a one-to-many relationship with the additional_test table. The propositionalization of the examination and additional test tables will lead to a new examination table completed with new features, such as a count of how many tests were bad.

Examination and its additional_tests

eid	pid	Date	Weight	Smokes	BP	…	Bad tests	…
1	I	10/10/1991	60	Yes	10	…	1	…
2	I	04/06/1992	64	Yes	12	…	0	…
⋮	⋮	⋮	⋮	⋮	⋮	…	⋮	…

Then the propositionalization of the patient table and the already propositionalized examination tables is performed, producing a new patient table completed with new features such as the mean value for each patient of the number of bad tests among all his or her examinations (f248)

Patient, his/her examinations and additional_tests

pid	name	…	f60	…	f248	…
I	Smith	…	2	…	1	…
⋮	⋮	…	⋮	…	⋮	…

It is not necessarily meaningful to aggregate at an intermediate level. An alternative is to join complementary tables first and apply the aggregation at the individual level only. A variant consists in replacing the join by a propagation of the identifier, i.e., adding the identifier of the individual into all related tables. Both lead to a kind of "star schema" where the individual is directly linked to all complementary tables.

Example 7 In our simplified medical domain, it is perhaps more interesting to first relate all additional tests to their patients, then aggregate on similar tests. First the complementary tables are joined

Additional_test and its examination

tid	Name	Value	Interpretation	eid	pid	Weight	…
t237	Red blood cells	35	Bad	1	I	60	…
t238	Radiography	Nothing	Good	1	I	60	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	…
t574	Red blood cells	43	Good	2	I	64	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	…

Let us emphasize the difference with the propositionalized examination and its additional_tests table of Example 6

There is a one-to-many relationship from the patient table to that new additional_test and its examination table. Aggregation functions can be used to build features such as the minimum percentage of red blood cells (f352)

Patient, his/her additional_tests and examinations

pid	Name	…	f60	…	f352	…
I	Smith	…	2	…	35	…
⋮	⋮	…	⋮	…	⋮	…

Finally, different propositionalization approaches can be combined, by a simple join.

Future Directions

Propositionalization explicitly aims at leaving attribute selection to the propositional learner applied afterward. The number of potential features is large. No existing propositionalization system is able to enumerate all imaginable features. Historically existing approaches have focused on a subset of potential features, e.g., numerical aggregation functions without selection (Knobbe et al. 2001) and selection based on a single elementary condition and existential aggregation (Flach and Lachiche 1999; Kramer et al. 2001). Most approaches can be combined to provide more features. The propositionalization should be guided by the user.

Propositionalization is closely related to knowledge representation. Specific representational issues require appropriate propositionalization techniques, e.g., Perlich and Provost (2006) introduce new propositionalization operators to deal with high-cardinality categorical attributes. New data sources, such as geographical or multimedia data, will need an appropriate representation and perhaps appropriate propositionalization operators to apply off-the-shelf attribute-value learners.

Propositionalization raises three fundamental questions. The first question is related to knowledge representation. That question is whether the user should adapt to existing representations, and accept a need to propositionalize, or whether data can be mined from the data sources, requiring the algorithms to be adapted or invented. The second question is whether propositionalization is needed. Propositionalization explicitly allows the user to contribute to the feature elaboration and invites him or her to guide the search, thanks to that language bias. It separates feature elaboration from model extraction. Conversely, relational data mining techniques automate the elaboration of the relevant attributes during the model extraction, but at the same time leave less opportunity to select the features by hand.

The third issue is one of efficiency. A more expressive representation necessitates a more complex search. Relational learning algorithms face the same dilemma as attribute-value learning in the form of a choice between an intractable search in the complete search space and an ad hoc heuristic/search bias (cf. ▶ search bias). They only differ in the size of the search space (cf. ▶ hypothesis space). Propositionalization is concerned with generating the search space. Generating all potential features is usually impossible. So practitioners have to constrain the propositionalization, e.g., by choosing the aggregation functions, the complexity of the selections, etc.; by restricting the numbers of operations; and so on. Different operators fit different problems and might lead to differences in performance (Krogel et al. 2003).

Cross-References

▶ Attribute
▶ Feature Construction in Text Mining
▶ Feature Selection
▶ Inductive Logic Programming
▶ Language Bias

▶ Learning from Structured Data
▶ Multi-instance Learning
▶ Relational Learning
▶ Statistical Relational Learning

Recommended Reading

Dietterich TG, Lathrop RH, Lozano-Pérez T(1997) Solving the multiple-instance problem with axis-parallel rectangles. Artif Intell 89(1–2):31–71

Džeroski S, Lavrač N (eds) (2001) Relational data mining. Springer, New York

Flach P, Lachiche N (1999) 1BC: a first-order Bayesian classifier. In: Džeroski S, Flach P (eds) Proceedings of the ninth international workshop on inductive logic programming (ILP'99). Volume 1634 of lecture notes in computer science. Springer, pp 92–103

Knobbe AJ, de Haas M, Siebes A (2001) Propositionalisation and aggregates. In: Proceedings of the sixth European conference on principles of data mining and knowledge discovery. Volume 2168 of lecture notes in artificial intelligence. Springer, pp 277–288

Kramer S, Lavrač N, Flach P (2001) Propositionalization approaches to relational data mining. In: Džeroski S, Lavrač N (eds) Relational data mining. Springer, New York, chap 11, pp 262–291

Krogel M-A, Rawles S, Železný F, Flach PA, Lavrač N, Wrobel S (2003) Comparative evaluation of approaches to propositionalization. In: Horváth T, Yamamoto A (eds) Proceedings of the thirteenth international conference on inductive logic programming. volume 2835 of lecture notes in artificial intelligence. Springer, pp 197–214

Lachiche N (2005) Good and bad practices in propositionalisation. In: Bandini S, Manzoni S (eds) Proceedings of advances in artificial intelligence, ninth congress of the Italian association for artificial intelligence (AI*IA'05). Volume 3673 of lecture notes in computer science. Springer, pp 50–61

Perlich C, Provost F (2006) Distribution-based aggregation for relational learning with identifier attributes. Mach Learn 62:62–105

Srinivasan A, Muggleton S, King RD, Stenberg M (1996) Theories for mutagenicity: a study of first-order and feature based induction. Artif Intell 85(1–2):277–299

Tomečková M, Rauch J, Berka P (2002) Stulong – data from longitudinal study of atherosclerosis risk factors. In: Berka P (ed) Discovery challenge workshop notes, ECML/PKDD'02.

Prospective Evaluation

Prospective evaluation is an approach to ▶ Out-Of-Sample Evaluation whereby a model learned from historical data is evaluated by observing its performance on new data as they become available. Prospective evaluation is likely to provide a less biased estimation of future performance than evaluation on historical data.

Cross-References

▶ Algorithm Evaluation

Pruning

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt,
Darmstadt, Deutschland
Department of Information Technology,
University of Leoben, Leoben, Austria

Abstract

Pruning describes the idea of avoiding ▶ Overfitting by simplifying a learned concept, typically after the actual induction phase.

Method

The term originates from decision tree learning, where the idea of improving the decision tree by cutting some of its branches may be viewed as an analogy to the concept of pruning in gardening.

Commonly, one distinguishes two types of pruning:

Pre-pruning monitors the learning process and prevents further refinements if the current hypothesis becomes too complex.

Post-pruning first learns a possibly overfitting hypothesis and then tries to simplify it in a separate learning phase.

Pruning techniques are particularly important for state-of-the-art ▶ Decision Tree and ▶ Rule Learning algorithms (see there for more details).

The key idea of pruning is essentially the same as ▶ Regularization in statistical learning, with the key difference that regularization incorporates a complexity penalty directly into the learning heuristic, whereas pruning uses a separate pruning criterion or pruning algorithm.

Cross-References

▶ Decision Tree
▶ Regularization
▶ Rule Learning

Pruning Set

Definition

A pruning set is a subset of a ▶ training set containing data that are used by a learning system to evaluate models that are learned from a ▶ growing set.

Cross-References

▶ Data Set

Q

Q-Learning

Peter Stone
Department of Computer Science, The University of Texas at Austin, Austin, TX, USA

Abstract

Definition of Q-learning.

Definition

Q-learning is a form of ▶ temporal difference learning. As such, it is a model-free ▶ reinforcement learning method combining elements of ▶ dynamic programming with Monte Carlo estimation. Due in part to Watkins' (1989) proof that it converges to the optimal value function, Q-learning is among the most commonly used and well-known ▶ reinforcement learning algorithms.

Cross-References

▶ Reinforcement Learning
▶ Temporal Difference Learning

Recommended Reading

Watkins CJCH (1989) Learning from delayed rewards. PhD thesis. King's College, Cambridge

Quadratic Loss

▶ Mean Squared Error

Qualitative Attribute

▶ Categorical Attribute

Quality Threshold

▶ Quality Threshold Clustering

Quality Threshold Clustering

Xin Jin[1] and Jiawei Han[2]
[1]PayPal Inc., San Jose, CA, USA
[2]University of Illinois at Urbana-Champaign, Urbana, IL, USA

Abstract

Quality Threshold is a clustering algorithm without specifying the number of clusters. It uses the maximum cluster diameter as the parameter to control the quality of clusters.

Synonyms

Quality threshold

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

Definition

Quality Threshold (QT) clustering (Heyer et al. 1999) is a partitioning clustering algorithm originally proposed for gene clustering. The focus of the algorithm is to find clusters with guaranteed quality. Instead of specifying K, the number of clusters, QT uses the maximum cluster diameter as the parameter.

The basic idea of QT is as follows: form a candidate cluster by starting with a random point and iteratively add other points, with each iteration adding the point that minimizes the increase in cluster diameter. The process continues until no point can be added without surpassing the diameter threshold. If surpassing the threshold, a second candidate cluster is formed by starting with a point and repeating the procedure. In order to achieve reasonable clustering quality, already assigned points are available for forming another candidate cluster.

For data partition, QT selects the largest candidate cluster and removes the points which belong to the cluster from consideration and repeats the procedure on the remaining set of data.

The advantage of QT clustering is that it can guarantee cluster quality and does not require the prior knowledge of the cluster number. The disadvantage is that the algorithm is computationally expensive as much as $O(N^3)$.

Softwares

The following softwares have implementations of the Quality Threshold (QT) clustering algorithm:

- Flexclust: Flexible Cluster Algorithms. R package. http://cran.r-project.org/web/packages/flexclust/index.html
- FinMath. A numerical library that provides components for the development of mathematical, scientific, and financial applications on the .NET platform. https://www.rtmath.net

Recommended Reading

Heyer L, Kruglyak S, Yooseph S (1999) Exploring expression data: identification and analysis of coexpressed genes. Genome Res 9:1106–1115

Quantitative Attribute

▶ Numeric Attribute

Quantum Machine Learning

Maria Schuld[1] and Francesco Petruccione[2]
[1]Quantum Research Group, School of Chemistry & Physics, University of KwaZulu-Natal, Durban, South Africa
[2]National Institute of Theoretical Physics (NITheP), KwaZulu-Natal, South Africa

Abstract

Quantum machine learning is a young research area investigating which consequences the emerging technology of quantum computing has for machine learning. This article introduces into basic concepts of quantum information and summarises some major strategies of implementing machine learning algorithms on a quantum computer.

Definition

Quantum machine learning (QML) is a subdiscipline of quantum information processing research, with the goal of developing *quantum algorithms* that learn from data in order to improve existing methods in machine learning. A quantum algorithm is a routine that can be implemented on a *quantum computer*, a device that exploits the laws of quantum theory in order to process information.

A number of quantum algorithms have been proposed for various machine learning models such as neural networks, support vector machines, and graphical models, some of which claim runtimes that under certain conditions grow only logarithmic with the size of the input space and/or dataset compared to conventional methods. A crucial point for runtime considerations is to find a procedure that efficiently encodes classical data into the properties of a quantum system. QML algorithms are often based on well-known quantum subroutines (such as *quantum phase estimation* or *Grover search*) or exploit fast annealing techniques through quantum tunneling and can make use of an exponentially compact representation of data through the probabilistic description of quantum systems.

Besides finding quantum algorithms for pattern recognition and data mining, QML also investigates more fundamental questions about the concept of learning from the perspective of quantum theory. Sometimes the definition of QML is extended by research that applies machine learning *to* quantum information, such as is frequently done when the full evolution or state of a quantum system has to be reconstructed from limited experimental data.

Motivation and Background

The accurate solution of many learning problems is known to be NP-hard, such as the training of Boltzmann machines or inference in graphical models. But also methods for which tractable algorithms are known suffer from the increasing size of datasets available in today's applications. The idea behind QML is to approach these problems from the perspective of quantum information and harvest the power of quantum computers for applications in artificial intelligence and data mining.

The motivation to find quantum analogues for "classical" machine learning algorithms derives from the success of the dynamic research field of quantum information. Some speedups compared to the best or best-known classical algorithms have already been shown, the most prominent being Shor's factorization algorithm (Shor 1997) (providing an exponential speedup compared to the best classical algorithm known) and Grover's search algorithm for unsorted databases (Grover 1996) (providing a quadratic speedup to the best possible classical algorithm). Although it is still an open question whether "true" exponential speedups are possible, the number of quantum algorithms is constantly growing. Also the technological implementation of large-scale universal quantum computers makes steady progress, and many proof-of-principle experiments have confirmed the theoretical predictions (The reader can get a first impression of the current progress in Wikipedia's "timeline of quantum computing" https://en.wikipedia.org/wiki/Timeline_of_quantum_computing.) The first realizations of quantum annealing devices, which solve a very specific type of optimization problem and are thus not universal, are already commercially available (e.g., http://www.dwavesys.com/).

Proposals that apply quantum computing to data mining in general and learning tasks in particular have been sporadically put forward since quantum computing became a well-established research area in the 1990s. A specifically large share of attention has been devoted to so-called *quantum neural network* models which simulate the behavior of artificial neural networks based on quantum information. They were initially motivated by questions of whether quantum mechanics can help to explain the functioning of our brain (Kak 1995) and vary in the degree of a rigorous application of quantum theory (Schuld et al. 2015). Since around 2012, there has been a rapid increase in other contributions to QML, consisting of proposals for quantum versions of hidden Markov models (Barry et al. 2014), Boltzmann machines (Wiebe et al. 2014; Adachi and Henderson 2015), belief nets (Low et al. 2014), support vector machines (Rebentrost et al. 2014), linear regression (Schuld et al. 2016), Gaussian processes (Zhao et al. 2015) and many more. Several collaborations between IT companies and academic institutions have been created and promise to advance the field of QML in future. For example, Google and NASA founded the *Quantum Artificial Intelligence Lab* in 2013, the University of

Oxford and Nokia set up a *Quantum Optimisation and Machine Learning* program in 2015, and the University of Southern California collaborates with Lockheed Martin on machine learning applications through the *Quantum Computation Center*.

Quantum Computing

In order to present the major approaches to QML research below, it is necessary to introduce some basic concepts of quantum information. The interested reader shall be referred to the excellent introduction by Nielsen and Chuang (2010).

In conventional computers, the state of a physical system represents bits of information and is manipulated by the Newtonian laws of physics (e.g., the presence of a current in a circuit represents 0 and 1 and is manipulated by the laws of electrodynamics). A quantum computer follows a very similar concept, only that the underlying physical system is governed by the laws of quantum theory and is therefore called a *quantum system*.

Quantum theory is a mathematical apparatus describing physical objects on very small scales (i.e., electrons, atoms, photons). More precisely, it is a probabilistic description of the results of physical measurements on quantum systems, and although confirmed in many experiments, it shows a number of features distinct to classical or Newtonian mechanics. Quantum computers exploit these features through information processing based on the rules of quantum theory. Although a number of exciting results have been achieved, it is still unknown whether BQP, the class of decision problems solvable by a quantum computer in polynomial time, is larger than BPP, its classical analogue. In short, quantum computing is a very dynamic research area with many promising results and open questions.

The quantum information community uses a variety of computational models that have been shown to be equivalent, but which constitute different building blocks of universal quantum computation. The following will give a short introduction to the most influential model, the *circuit model*, to clarify important concepts on which QML algorithms are based.

The Concept of a Qubit

A central concept in the major quantum computational models is the *qubit*, an abstraction of a quantum system that has two possible configurations or states. As long as certain properties are fulfilled (DiVincenzo 2000), such a *two-level system* can have many possible physical realizations (just like bits may be encoded in currents of circuits or the pits and lands of CDs), for example, a hydrogen atom in the energetic ground or first excited state, the current in a superconducting circuit or the path a light photon chooses through a semitransparent mirror.

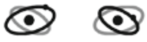
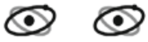
Qubits are often introduced as "bits that can be in states 0 and 1 at the same time," which mystifies rather than explains the concept. In fact, qubits can be compared to a probabilistic description of a *classical* physical system with two different states, say a coin with the states "heads" and "tails." As illustrated in Table 1, the probabilities p_{00}, p_{01}, p_{10}, and p_{11} with $\sum_i p_i = 1$ describe our expectation to get the respective result "head and head," "head and tail," "tail and head," and "tail and tail" after tossing two coins. Note that the coin tosses do not necessarily need to be statistically independent events.

The probabilistic description of a qubit shows a significant difference (see Table 2). The four configurations "00," "01," "10," and "11" of a two-qubit system such as two simplified atoms

Quantum Machine Learning, Table 1 Probabilistic description of a classical system of two coins. Each of the four possible outcomes or configurations after tossing both coins is associated with a probability.

example	config.	probability
① ①	'00'	p_{00}
● ①	'01'	p_{01}
● ①	'10'	p_{10}
● ●	'11'	p_{11}

Quantum Machine Learning, Table 2 Important elements in the description of a two-qubit system. An example is two atoms that can each be in the ground and first excited state, so that the system has four possible abstract configurations. Quantum theory associates each configuration (or potential measurement outcome) with an *amplitude*, and the absolute square of the amplitude is the probability of measuring this state. In the mathematical notation, each configuration corresponds to a unit basis vector or, in Dirac notation, a Dirac basis state

example	config.	probability	amplitude	unit basis	Dirac basis
	00	$\|a_{00}\|^2$	a_{00}	$(1,0,0,0)^T$	$\|00\rangle$
	01	$\|a_{01}\|^2$	a_{01}	$(0,1,0,0)^T$	$\|01\rangle$
	10	$\|a_{10}\|^2$	a_{10}	$(0,0,1,0)^T$	$\|10\rangle$
	11	$\|a_{11}\|^2$	a_{11}	$(0,0,0,1)^T$	$\|11\rangle$

are each associated with a complex number called *amplitude*, and the probability of observing the two qubits in one of the four possible joint states is given by the absolute square of the amplitude. The sum of absolute squares of the amplitudes $a_i, i = 1, \ldots, 2^n$ of an n-qubit system consequently has to add up to one, $\sum_i |a_i|^2 = 1$. In both the classical and the quantum case, once the coins or atoms are observed in one of the joint configurations, their state is fully determined, and repeated observations will only confirm the result. As will be explained below, this concept of complex amplitudes is central to quantum information and has up to the present – 100 years after the beginning of quantum theory – still not found a satisfying interpretation for our everyday intuition.

Algorithmic Manipulations of Qubits

Information processing is about the manipulation of bits by elementary logic gates such as AND or XOR, and quantum information processing likewise needs to define elementary operations on qubit systems (of course derived from the laws of quantum theory), from which algorithms with a well-defined output can be constructed.

In a probabilistic description, manipulating information corresponds to a transformation of the system's probability distribution. For example, in the case of the two coins, this could mean drawing a "heads" over the "tails" symbol, causing the coin to only toss "heads." Using the mathematical language of Markov chains, changes of a *classical* probability distribution can be expressed by a linear transformation applied to the vector of probabilities, written as a stochastic matrix $S = (s_{ij})$ multiplied from the left. The stochastic matrix has the properties that its entries are nonnegative and all columns sum up to one, in order to guarantee that the resulting vector on the right side is again a probability distribution. In our two-coin example, this reads

$$S \begin{pmatrix} p_{00} \\ p_{01} \\ p_{10} \\ p_{11} \end{pmatrix} = \begin{pmatrix} p'_{00} \\ p'_{01} \\ p'_{10} \\ p'_{11} \end{pmatrix}, \qquad \begin{array}{c} s_{ij} \geq 0, \\ \sum_i s_{ij} = 1. \end{array} \tag{1}$$

For *quantum* systems, any physically possible evolution can be mathematically represented by a unitary matrix $U = (u_{ij})$ applied to the vector of amplitudes, which in the two-qubit example reads

$$U \begin{pmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{pmatrix} = \begin{pmatrix} a'_{00} \\ a'_{01} \\ a'_{10} \\ a'_{11} \end{pmatrix}, \qquad \begin{array}{c} u_{ij} \in \mathbb{C}, \\ S^\dagger S = 1. \end{array} \tag{2}$$

A unitary matrix has orthogonal column vectors, guaranteeing that the resulting vector on the right side is again a quantum amplitude vector. Equation (2) describes in fact any possible closed evolution of a two-qubit system in quantum theory.

Quantum algorithms (as well as QML algorithms) are usually formulated using the Dirac notation, in which one decomposes the amplitude vector into a linear combination of unit vectors and rewrites the unit vectors as Dirac vectors:

$$\mathbf{a} = a_1 \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \ldots + a_{2^n} \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} \tag{3}$$

$$\Updownarrow \tag{4}$$

$$|\psi\rangle = a_1 \, |0 \ldots 0\rangle + \ldots + a_{2^n} \, |1 \ldots 1\rangle \, . \tag{5}$$

Dirac notation is very handy as it visualizes the actual measurement result of the n qubits corresponding to an amplitude.

Similarly to elementary gates, the circuit model defines elementary unitary transformations as building blocks to manipulate the quantum state of a qubit system. For example, consider a single qubit described by the complex amplitude vector $(a_1, a_2)^T$. If the quantum system is in state $(1, 0)^T$, we know with certainty that a measurement will produce the state 0 (since the probability of measuring the 0 state is given by $p_0 = |a_1|^2 = 1.0$, while $p_1 = |a_2|^2 = 0.0$). The unitary transformation

$$U_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

then transforms this state into $(0, 1)^T$, which will certainly result in a measurement of state 1. U_x hence effectively performs a bit flip or NOT gate on the state of the qubits. In a similar fashion, other quantum gates can be defined that together form a set of universal gates for quantum computation.

Why Is Quantum Computing Different?

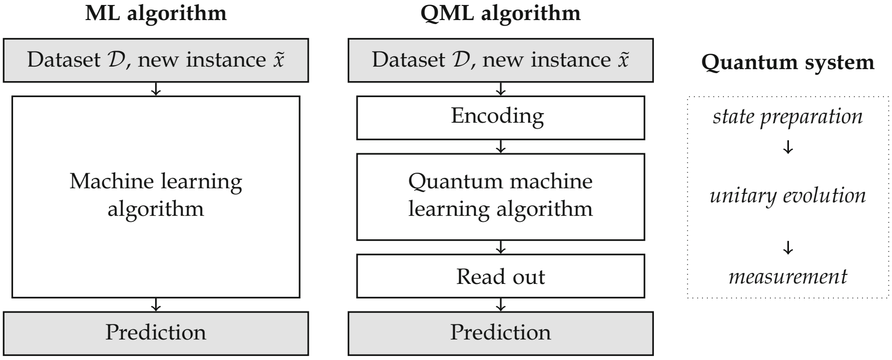
Returning to the question why complex amplitudes change the rules of classical information processing, consider another elementary quantum gate that has no classical equivalent since it cannot be expressed as a stochastic matrix with positive entries. The Hadamard gate

$$U_H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} ,$$

will, applied to a state $(1, 0)^T$, produce $(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})^T$, which is called a *superposition* of states 0 and 1. A classical equivalent would be a state of maximum uncertainty, as the probability of measuring the qubit in state 0 or 1 is $|\frac{1}{\sqrt{2}}|^2 = \frac{1}{2}$ each. However, the difference of a superposition becomes apparent when applying U_H once more, which transforms the state back into $(1, 0)^T$ as the minus in U_H cancels the two amplitudes with each other when calculating the second entry of the resulting amplitude vector. In other words, amplitudes can annihilate each other, a phenomenon called *interference* which is often mentioned as the crucial resource of quantum computing. Beyond this illustration, the elegant theory of quantum Turing machines allows a more sophisticated comparison between quantum and classical computing (Deutsch 1985), but goes beyond our scope here.

Quantum Machine Learning Algorithms

Most existing QML algorithms solve problems of supervised or unsupervised pattern classification and regression, although first advancements to reinforcement learning have been made (e.g., Paparo et al. 2014). Given a (classical) dataset $\mathcal{D}$ and a new instance $\tilde{x}$ for which we would make a prediction, a QML algorithm usually consists of three parts: First, the input has to be encoded into a quantum system through a *state preparation* routine. Second, the quantum algorithm is executed by unitary transformations (Note that nonunitary evolutions are possible in so-called *open quantum systems*, but correspond to a unitary evolution of a larger system.) Third, the result is read out by measuring the quantum system (see Fig. 1). The encoding and readout steps are often the bottlenecks of a QML algorithm; for example, reading out an amplitude in a quantum state that is in a uniform superposition of all possibilities will on average take a number of measurements that is exponential in the number of qubits. In particular, claims of quantum algorithms that run in time logarithmic in the size of the dataset and input vectors often ignore the resources it takes for the crucial step of encoding the information carried by a dataset into


ML algorithm
Dataset $\mathcal{D}$, new instance $\tilde{x}$
Machine learning algorithm
Prediction
QML algorithm
Dataset $\mathcal{D}$, new instance $\tilde{x}$
Encoding
Quantum machine learning algorithm
Read out
Prediction
Quantum system
state preparation
unitary evolution
measurement

Quantum Machine Learning, Fig. 1 Comparison of the basic scheme of classical (*left*) and quantum (*center*) machine learning algorithms for pattern classification, together with the operations on the quantum system (*right*). In order to solve machine learning tasks based on classical datasets, the quantum algorithm requires an information encoding and readout step that are in general highly nontrivial procedures, and it is important to consider them in the runtime

a quantum system. Such algorithms can still be valuable for pure quantum information processing, i.e., if the "quantum data" is generated by previous routines or experiments.

The QML algorithm and readout step depend heavily on the way information is encoded into the quantum system; one can distinguish three ways of information encoding into an n-qubit system:

1. Interpreting the possible measurement outcomes of a qubit system as a bit sequence.
2. Interpreting the amplitude vector as (i) a 2^n-dimensional classical real vector or (ii) a probability distribution over n binary variables.
3. Encoding the result to an optimization problem into the ground state (state of the lowest energy) of a quantum system.

These strategies help to distinguish different approaches to develop QML algorithms.

Associating Qubits with Bits

The most straightforward method of information encoding into quantum systems is to associate bits with qubits. For example, the two-qubit state $(1, 0, 0, 0)^T$ in the example in Table 2 represents the bit string [00], since the system has unit probability of being measured in the '00' state.

To encode a full dataset in this fashion, it needs to be given in binary form, meaning that every feature vector (and, if applicable, its label) has been translated into an n-bit binary sequence. For example, the dataset $\mathcal{D}$ [$\mathcal{D} = \left\{ \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \right\}$] can be encoded into the quantum state $\mathbf{a}_{\mathcal{D}} = \frac{1}{\sqrt{3}}(01010010)^T$:

In this case, the Dirac notation introduced above is helpful as it explicitly contains the encoded feature vectors:

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{3}}(|001\rangle + |011\rangle + |110\rangle).$$

An early example of a QML algorithm based on such a "quantum dataset" has been developed for pattern completion (finding feature vectors containing a given bit sequence) by an associative memory mechanism as known from Hopfield models (Ventura and Martinez 2000). The authors suggest a routine to construct the state $\mathbf{a}_{\mathcal{D}}$ efficiently and use a modified Grover search algorithm, in which the amplitudes corresponding to the desired measurement outcomes are marked in one single step, after which the amplitudes of the marked states are amplified. The resulting quantum state has a high probability of being measured in one of the basis states containing the desired bit sequence.

An example of a QML algorithm for supervised pattern classification is a quantum version of k-nearest neighbor (Schuld et al. 2014b). Beginning with a superposition as in Eq. (6) where some selected qubits encode the class label, the idea is to weigh the amplitudes by the Hamming distance between each corresponding training vector and the new input. Only the "class-label qubits" get measured, so that close inputs contribute more to the probability of measuring their class label than distant ones. An alternative is presented by Wiebe et al. (2015), who also prepare a quantum state with distance-weighted amplitudes and then performed a subroutine based on the Grover search to find the basis state representing the closest neighbor.

Encoding Information into Amplitudes

Another way to encode information is to associate the quantum amplitude vector with a real classical vector:

$$\begin{pmatrix} a_1 \\ \vdots \\ a_{2^n} \end{pmatrix} \leftrightarrow \begin{pmatrix} x_1 \\ \vdots \\ x_{2^n} \end{pmatrix}, \sum_i |x_i|^2 = 1, x_i \in \mathbb{R}.$$

Note that since amplitude vectors are normalized, the classical vector has to be preprocessed accordingly. A quantum system of n qubits can therefore in principle encode 2^n real numbers, which is an exponentially compact representation. There are some vectors for which state preparation can be done in time that grows only linear with the number of qubits, and if the QML algorithm and readout step have the same property, an algorithm which is logarithmic in the input dimension is found.

Two different strategies to use this encoding for QML can be distinguished, one that associates the amplitude vector with one or all feature vectors in order to use the power of eigenvalue decomposition inherent in the formalism of quantum theory, and the other in which amplitudes are used to encode classical probability distributions.

Quantum Eigenvalue Decompositions

An important branch of QML research is based on the intrinsic feature of quantum theory to evaluate eigenvalues of operators, which has been exploited in an important quantum algorithm for the solution of systems of linear equations (Harrow et al. 2009). The routine takes a quantum state described by the amplitude vector $\mathbf{b}$ which corresponds to the (normalized) right side of a classical linear system of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$. Through a set of involved operations (including the Hamiltonian simulation of an operator corresponding to $\mathbf{A}$, a quantum phase estimation algorithm and a selective measurement that has to be repeated until a certain result was obtained), the quantum state is transformed into $\sum_j \lambda_j^{-1} \mathbf{u}_j^T \mathbf{b}\, \mathbf{u}_j$ with eigenvalues λ_j and eigenvectors $\mathbf{u}_j$ of $\mathbf{A}$, which equals the correct solution $\mathbf{x}$. Due to the exponentially compact representation of information, the complexity of the algorithm depends only logarithmically on the size of $\mathbf{b}$ when we ignore the encoding and readout step. However, its running time depends sensibly on other parameters such as the condition number and sparsity of $\mathbf{A}$, as well as the desired accuracy in the result. This makes the linear systems algorithm only applicable to very special problems (Aaronson 2015). QML researchers have tried to find such applications in different areas of machine learning that rely on matrix inversion.

The first full QML example exploiting the ideas of the linear systems algorithm was the quantum support vector machine (Rebentrost et al. 2014). The main idea is to take the dual formulation of support vector machines written as a least squares problem, in which a linear system of equations with the kernel matrix $K_{ij} = \mathbf{x}_i \mathbf{x}_j$ and $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}$ has to be solved, and apply the above quantum routine. By making use of a trick, the linear systems algorithm can take a quantum state encoding K_{ij} (instead of a quantum operator as in the original version). Creating a quantum version of K_{ij} is surprisingly elegant if one can prepare a quantum state:

$$(x_1^1, \ldots, x_N^1, \ldots, x_1^M, \ldots, x_N^M) \qquad (6)$$

whose amplitudes encode the MN features of all training vectors $\mathbf{x}^m = (x_1^m, \ldots, x_N^m)^T$ $m = 1, \ldots, M$. The statistics of a specific subset of the qubits in state Eq. (6) include a covariance matrix (in quantum theory known as density matrix) that is entrywise equivalent to the kernel and which can be accessed by further processing.

Data fitting by linear regression has been approached by means of the quantum linear systems algorithm by Wiebe et al. (2012) to obtain the well-known least squares solution:

$$\mathbf{w} = \mathbf{X}^{+}\mathbf{y}$$

for the linear regression parameters $\mathbf{w}$ with the pseudoinverse $\mathbf{X}^{+} = (\mathbf{X}^{+}\mathbf{X})^{-1}\mathbf{X}^{+}$ where the columns of X are the training inputs. Schuld et al. (2016), propose another version of the quantum algorithm that is suited for prediction. The algorithm is based on a quantum computation of the singular value decomposition of $\mathbf{X}^{+}$ which in the end encodes the result of the prediction of a new input into the measurement result of a single qubit.

Other QML algorithms based on the principle of matrix inversion and eigenvalue estimation on a quantum computer have been proposed for Gaussian processes (Zhao et al. 2015) as well as to find topological and geometric features of data (Lloyd et al. 2016). The routines discussed here specify the core algorithm as well as the readout step in the scheme of Fig. 1 and are logarithmic in the dimension of the feature vectors. However, they leave the crucial encoding step open, which might merely "hide" the complexity for all but some selected problems as has been critically remarked by Aaronson (2015).

Quantum Probability Distributions

Since quantum theory defines probability distributions over measurement results, it is immediately apparent that probability distributions over binary variables can very genuinely be represented by the amplitudes of a qubit system.

More precisely, given n random binary variables, an amplitude vector can be used to encode the square roots of 2^n probabilities of the different realizations of these variables. For example, the probability distribution over the possible results of the two-coin toss in Table 1 could be encoded into an amplitude vector $\left(\sqrt{p_{00}}, \sqrt{p_{01}}, \sqrt{p_{10}}, \sqrt{p_{11}}\right)$ of the two-qubit system in Table 2. Despite the efficient representation of probability distributions, also the marginalization of variables, which is intractable in classical models, corresponds to the simple step of excluding the qubits corresponding to these variables from measurements and considering the resulting statistics.

While these advantages sound impressive, it turns out that the problem of statistical inference remains prohibitive: Conditioning the qubit probability distribution on the state of all but one qubit, $p(x_1, \ldots, x_N) \rightarrow p(x_N|x_1, \ldots, x_{N-1})$, requires measuring these qubits in exactly the desired state, which has in general an exponentially small probability. Measuring the state can be understood as sampling from the probability distribution, and one has to do an unfeasibly large number of measurements to obtain the conditional statistics, while after each measurement a new quantum state has to be prepared. It has in fact been shown that the related problem of Bayesian updating through quantum distribution is intractable (Wiebe and Granade 2015), as it corresponds to a Grover search which can only be quadratically faster than classically possible.

Even without the ability for efficient inference, quantum systems can still be interesting for probabilistic machine learning models. Low et al. (2014) exploit the quadratic speedup for a problem of inference with "quantum Bayesian nets." Hidden Markov models have been shown to have an elegant formal generalization in the language of open quantum systems (Barry et al. 2014). Wiebe et al. (2014) show how quantum states that approximate Boltzmann distributions can be prepared to get samples for the training of Boltzmann machines through contrastive divergence. The same authors propose a semi-classical routine for Bayesian updating (Wiebe and Granade 2015). These contributions suggest that a lot of potential lies in approaches that exploit the genuinely stochastic structure of quantum theory for probabilistic machine learning methods.

Optimization and Quantum Annealing

Another branch of QML research is based on techniques of *quantum annealing*, which can be understood as an analogue version of quantum computing (Das and Chakrabarti 2008). Similar to the metaheuristic of simulated annealing, the idea is to drive a physical system into its energetic ground state which encodes the desired result of an optimization problem. To associate each basis state of a qubit with an energy, one has to introduce externally controllable physical interactions between the qubits.

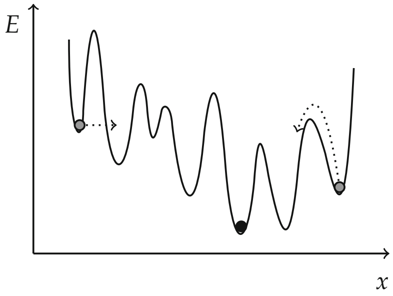
The main difference between classical and quantum annealing is that "thermal fluctuations" are replaced by quantum fluctuations which enable the system to *tunnel* through high and thin energy barriers (the probability of quantum tunneling decreases exponentially with the barrier width, but is independent of its height). That makes quantum annealing especially fit for problems with a "sharply ragged" objective function (see Fig. 2). Quantum annealing can be understood as a heuristic version of the famous computational model of quantum adiabatic computation, which is why some authors speak of *adiabatic quantum machine learning*.

The significance of quantum annealing lies in its relatively simple technological implementation, and quantum annealing devices are available commercially. Current machines are limited to solving quadratic unconstrained binary optimization (QUBO) problems:

$$\underset{(x_1,\ldots,x_N)}{\text{argmin}} \sum_{ij} w_{ij}\, x_i x_j \quad \text{with} \quad x_i, x_j \in [0,1]. \tag{7}$$

An important step is therefore to translate the problem into QUBO form, which has been done for simple binary classifiers or perceptrons (Pudenz and Lidar 2013; Denchev et al. 2012), image matching problems (Neven et al. 2008) and Bayesian network structure learning (O'Gorman et al. 2015). Other machine learning models naturally relate to the form of Eq. (7). For example, a number of contributions investigate quantum annealing for the sampling step required in the training of Boltzmann machines via contrastive divergence (Adachi and Henderson 2015; Amin et al. 2016). Another example is the Hopfield model for pattern recognition via associative memory, which has been investigated from the perspective of adiabatic quantum computation with nuclear magnetic resonance systems (Neigovzen et al. 2009).

Measuring the performance of quantum annealing compared to classical annealing schemes is a non-trivial problem, and although advantages of the quantum schemes have been demonstrated in the literature mentioned above, general statements about speedups are still controversial.



Quantum Machine Learning, Fig. 2 Illustration of quantum annealing in an energy landscape over (here continuous) states or configurations x. The ground state is the configuration of the lowest energy (*black dot*). Quantum tunneling allows the system state to transgress high and thin energy barriers (*gray dot* on the *left*), while in classical annealing technique stochastic fluctuations have to be large enough to allow for jumps over peaks (*gray dot* on the *right*)

Experimental Realizations

The reason why one rarely finds classical computer simulations of quantum machine learning algorithms in the literature is that the description of quantum systems is classically intractable due to the exponential size of the amplitude vectors. Until a large-scale universal quantum computer is built, only QML algorithms based on quantum annealing can be tested on real devices and benchmarked against classical machine learning algorithms. Some proof-of-principle experiments have nevertheless implemented few-qubit examples of proposed QML algorithms in the lab. Among those are experimental realizations of the quantum support vector machine (Cai et al. 2015) as well as quantum clustering algorithms (Li et al. 2015; Neigovzen et al. 2009).

Further Reading

The interested reader may be referred to existing reviews on quantum machine learning research (Schuld et al. 2014a, 2015; Adcock et al. 2015).

Recommended Reading

Aaronson S (2015) Read the fine print. Nat Phys 11(4):291–293

Adachi SH, Henderson MP (2015) Application of quantum annealing to training of deep neural networks. arXiv preprint arXiv:1510.06356

Adcock J, Allen E, Day M, Frick S, Hinchliff J, Johnson M, Morley-Short S, Pallister S, Price A, Stanisic S (2015) Advances in quantum machine learning. arXiv preprint arXiv:1512.02900

Amin MH, Andriyash E, Rolfe J, Kulchytskyy B, Melko R (2016) Quantum boltzmann machine. arXiv preprint arXiv:1601.02036

Barry J, Barry DT, Aaronson S (2014) Quantum partially observable markov decision processes. Phys Rev A 90:032311

Cai X-D, Wu D, Su Z-E, Chen M-C, Wang X-L, Li L, Liu N-L, Lu C-Y, Pan J-W (2015) Entanglement-based machine learning on a quantum computer. Phys Rev Lett 114(11):110504

Das A, Chakrabarti BK (2008) Colloquium: quantum annealing and analog quantum computation. Rev Mod Phys 80(3):1061

Denchev V, Ding N, Neven H, Vishwanathan S (2012) Robust classification with adiabatic quantum optimization. In: Proceedings of the 29th international conference on machine learning (ICML-12), Edinburgh, pp 863–870

Deutsch D (1985) Quantum theory, the church-turing principle and the universal quantum computer. Proc R Soc Lond A: Math Phys Eng Sci 400:97–117. The Royal Society

DiVincenzo DP (2000) The physical implementation of quantum computation. Fortschritte der Physik 48(9–11):771–783 ISSN 1521–3978

Grover LK (1996) A fast quantum mechanical algorithm for database search. In: Proceedings of the twenty-eighth annual ACM symposium on theory of computing. ACM, New York, pp 212–219

Harrow AW, Hassidim A, Lloyd S (2009) Quantum algorithm for linear systems of equations. Phys Rev Lett 103(15):150502

Kak SC (1995) Quantum neural computing. Adv Imaging Electron Phys 94:259–313

Li Z, Liu X, Xu N, Du J (2015) Experimental realization of a quantum support vector machine. Phys Rev Lett 114(14):140504

Lloyd S, Garnerone S, Zanardi P (2016) Quantum algorithms for topological and geometric analysis of data. Nat Commun 7:10138

Low GH, Yoder TJ, Chuang IL (2014) Quantum inference on Bayesian networks. Phys Rev A 89:062315

Neigovzen R, Neves JL, Sollacher R, Glaser SJ (2009) Quantum pattern recognition with liquid-state nuclear magnetic resonance. Phys Rev A 79(4):042321

Neven H, Rose G, Macready WG (2008) Image recognition with an adiabatic quantum computer i. Mapping to quadratic unconstrained binary optimization. arXiv preprint arXiv:0804.4457

Nielsen MA, Chuang IL (2010) Quantum computation and quantum information. Cambridge University Press, Cambridge

O'Gorman B, Babbush R, Perdomo-Ortiz A, Aspuru-Guzik A, Smelyanskiy V (2015) Bayesian network structure learning using quantum annealing. Eur Phys J Spec Top 224(1):163–188

Paparo GD, Dunjko V, Makmal A, Martin-Delgado MA, Briegel HJ (2014) Quantum speedup for active learning agents. Phys Rev X 4(3):031002

Rebentrost P, Mohseni M, Lloyd S (2014) Quantum support vector machine for big data classification. Phys Rev Lett 113:130503

Schuld M, Sinayskiy I, Petruccione F (2014a) The quest for a quantum neural network. Q Inf Process 13 (11):2567–2586

Schuld M, Sinayskiy I, Petruccione F (2014b) Quantum computing for pattern classification. Pham, Duc-Nghia, Park, Seong-Bae (Eds.) Springer International Publishing In: Lecture notes in computer science, vol 8862. Springer, pp 208–220

Schuld M, Sinayskiy I, Petruccione F (2015) Introduction to quantum machine learning. Contemp Phys 56(2):172–185

Schuld M, Sinayskiy I, Petruccione F (2016) Prediction by linear regression on a quantum computer. Phys Rev A 94(2):022342

Shor PW (1997) Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J Comput 26(5):1484–1509

Ventura D, Martinez T (2000) Quantum associative memory. Inf Sci 124(1):273–296

Wiebe N, Granade C (2015) Can small quantum systems learn? arXiv preprint arXiv:1512.03145

Wiebe N, Braun D, Lloyd S (2012) Quantum algorithm for data fitting. Phys Rev Lett 109(5):050505

Wiebe N, Kapoor A, Svore K (2014) Quantum deep learning. arXiv: 1412.3489v1

Wiebe N, Kapoor A, Svore K (2015) Quantum nearest-neighbor algorithms for machine learning. Q Inf Comput 15:0318–0358

Zhao Z, Fitzsimons JK, Fitzsimons JF (2015) Quantum assisted Gaussian process regression. arXiv preprint arXiv:1512.03929

Quasi-Interpolation

► Radial Basis Function Networks

Query-Based Learning

Sanjay Jain[1] and Frank Stephan[2]
[1]School of Computing, National University of Singapore, Singapore, Singapore
[2]Department of Mathematics, National University of Singapore, Singapore, Singapore

Abstract

Query learning models the learning process as a dialogue between a pupil (learner) and a teacher; the learner has to figure out the target concept by asking questions of certain types and whenever the teacher answers these questions correctly, the learner has to learn within the given complexity bounds. Complexity can be measured by both, the number of queries as well as the computational complexity of the learner. Query learning has close connections to statistical models like PAC learning.

Definition

Most learning scenarios consider learning as a relatively passive process where the learner observes more and more data and eventually formulates a hypothesis that explains the data observed. Query-based learning is an ▶ active learning process where the learner has a dialogue with a teacher, which provides on request useful information about the concept to be learned.

Detail

This article will mainly focus on query-based learning of finite classes and of parameterized families of finite classes. In some cases, an infinite class has to be learned where then the behavior of the learner is measured in terms of a parameter belonging to the concept. For example, when learning the class of all singletons $\{x\}$ with $x \in \{0,1\}^*$, the parameter would be the length n of x, and an algorithm based on membership queries would need up to $2^n - 1$ queries of the form "Is y in L?" to learn an unknown set $L = \{x\}$ with $x \in \{0,1\}^n$. In Query-based learning, the questions asked are similar to the following: Which classes can be learned using queries of this or that type? If queries of a given type are used to learn a parameterized class $\bigcup C_n$, is it possible to make a learner which (with or without knowledge of n) succeeds to learn every $L \in C_n$ with a number of queries that is polynomial in n? What is the exact bound on queries needed to learn a finite class C in dependence of the topology of C and the cardinality of C? If a query-based learner using polynomially many queries exists for a parameterized class $\bigcup C_n$, can this learner also be implemented such that it is computable in polynomial time?

In the following, let C be the class of concepts to be learned and the concepts $L \in C$ are subsets of some basic set X. Now the learning process is a dialogue between a learner and a teacher in order to identify a language $L \in C$, which is known to the teacher but not to the learner. The dialogue goes in turns and follows a specific protocol that goes over a finite number of rounds. Each round consists of a query placed by the learner to the teacher and the answer of the teacher to this query. The query and the answer have to follow a specific format (see Table 1) and there are the following common types, where $a \in X$ and $H \in C$ are data items and concepts chosen by the learner and $b \in X$ is a counterexample chosen by the teacher:

While for subset queries and superset queries it is not required by all authors that the teacher provides a counterexample in the case that the answer is "no," this requirement is quite standard for the case of equivalence queries. Without counterexamples, a learner would not have any real benefit from these queries in settings where faster convergence is required, than by just checking "Is $H_0 = L$?," "Is $H_1 = L$?," "Is $H_2 = L$?," . . ., which would be some trivial kind of algorithm.

Here is an example: Given the class C of all finite subsets of $\{0,1\}^*$, a learner using superset queries could just work as given in Table 2 to learn each set of the form $L = \{x_1, x_2, \ldots, x_n\}$ with $n + 1$ queries.

Query-Based Learning, Table 1 Types of Queries

Query name	Precise Query	Answer if true	Answer if false
Membership query	Is $a \in L$?	"Yes"	"No"
Equivalence query	Is $H = L$?	"Yes"	"No" plus b (where $b \in H - L \cup L - H$)
Subset query	Is $H \subseteq L$?	"Yes"	"No" plus b (where $b \in H - L$)
Superset query	Is $H \supseteq L$?	"Yes"	"No" plus b (where $b \in L - H$)
Disjointness query	Is $H \cap L = \emptyset$?	"Yes"	"No" plus b (where $b \in H \cap L$)

Query-Based Learning, Table 2 Learning finite sets using superset queries

Round	Query	Answer	Counterexample
1	Is $L \subseteq \emptyset$?	"No"	x_1
2	Is $L \subseteq \{x_1\}$?	"No"	x_2
3	Is $L \subseteq \{x_1, x_2\}$?	"No"	x_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	Is $L \subseteq \{x_1, x_2, \ldots, x_{n-1}\}$?	"No"	x_n
$n+1$	Is $L \subseteq \{x_1, x_2, \ldots, x_{n-1}, x_n\}$?	"Yes"	—

Here, of course, the order on how the counterexamples come up does not matter; the given order was just preserved for the reader's convenience. Note that the same algorithm works also with equivalence queries in place of superset queries. In both cases, the algorithm stops with outputting "$L = \{x_1, x_2, \ldots, x_n\}$" after the last query. However, the given class is not learnable using membership and subset queries which can be seen as follows: Assume that such a learner learns $\emptyset$ using the subset queries "Is $H_0 \subseteq L$?," "Is $H_1 \subseteq L$?," "Is $H_2 \subseteq L$?," ..., "Is $H_m \subseteq L$?" and the membership queries "Is $y_0 \in L$?," "Is $y_1 \in L$?," "Is $y_2 \in L$?," ..., "Is $y_k \in L$?" Furthermore, let D be the set of all counterexamples provided by the learner to subset queries. Now let $E = D \cup H_0 \cup H_1 \cup \ldots \cup H_m \cup \{y_0, y_1, \ldots, y_k\}$. Note that E is a finite set and let x be an element of $\{0,1\}^* - E$. If $L = \{x\}$, then the answers to these queries are the same to the case that $L = \emptyset$. Hence, the learner cannot distinguish between the sets $\emptyset$ and $\{x\}$; therefore, the learner is incorrect on at least one of these sets.

In the case that C is finite, one could just ask what is the number of queries needed to determine the target L in the worst case. This depends on the types of queries permitted and also on the topology of the class C. For example, if C is the power set of $\{x_1, x_2, \ldots, x_n\}$, then n membership queries are enough; but if C is the set of all singleton sets $\{x\}$ with $x \in \{0,1\}^n$, then $2^n - 1$ membership queries are needed to learn the concept, although in both cases the cardinality of C is 2^n. One can do with $\log(|C|)$ many equivalence queries with counterexamples in the case that a class-comprising hypothesis space is permitted. For this, each conjecture H has that for all x, $H(x)$ follows the majority of those $L \in C$ which are consistent with all previous counterexamples. Then each counterexample would invalidate the majority of the still valid/consistent members of C and thus give the logarithmic bound.

Angluin (2004) provides a survey of the prior results on questions like how many queries are needed to learn a given finite class. Maass and Turán (1992) showed that usage of membership queries in addition to equivalence queries does not speed up learning too much compared to the case of using equivalence queries alone. If EQ is the number of queries needed to learn C from equivalence queries alone (with counterexamples) and EMQ is the number of queries needed to learn C with equivalence queries and membership queries, then

$$\frac{EQ}{\log(EQ+1)} \leq EMQ \leq EQ;$$

here the logarithm is base 2. This result is based on a result of Littlestone (1988) who characterized the number of queries needed to learn from equivalence queries alone and provided a "standard optimal algorithm" for this task. Note that these two results used class-comprising hypothesis spaces, where one can make an equivalence query with a hypothesis which is not in the class to be learned – this technique permits to get meaningful counterexample.

Angluin (1987) showed that the class of all regular languages can be learned in polynomial time using queries and counterexamples. Here the learning time is measured in terms of two parameters: the number n of states that the smallest deterministic finite automaton generating the language has and the number m of symbols in the longest counterexample provided by the teacher. Ibarra and Jiang (1988) showed that the algorithm can be improved to need at most dn^3 equivalence queries when the teacher always returns the shortest counterexample; Birkendorf et al. (2000) improved the bound to dn^2. In these bounds, d is the size of the alphabet used for defining the regular languages to be learned.

Much attention has been paid to the following question: Which classes of Boolean formulas over n variables can be learned with polynomially many queries, uniformly in n (see, e.g., Aizenstein et al. 1992; Aizenstein and Pitt 1995; Angluin et al. 1993; Hellerstein et al. 1996)? Angluin et al. (1993) showed that read-once formulas, in which every variable occurs only once, are learnable in polynomial time using membership and equivalence queries. On the other hand, read-thrice DNF (disjunctive normal form) formulas cannot be learned in polynomial time using the same queries (Aizenstein et al. 1992) unless $P = NP$. In other words, such a learner would not succeed because of the limited computational power of a polynomial time learner; hence, equipping the learner with an additional oracle that can provide this power would permit to build such a learner. Here an oracle – in contrast to a teacher – does not know the task to be learned but gives information which is difficult or impossible to compute. Such an oracle could, for example, be the set SAT of all satisfiable formulas, and thus the learner could gain additional power by asking the oracle whether certain formulas are satisfiable. A special class of Boolean formulas is that of Horn clauses (see, e.g., Angluin et al. 1992; Arias 2004; Arias and Balcázar 2009; Arias and Khardon 2002).

There are links to other fields. Angluin (1988, 1990) investigated the relation between query learning and ▶ PAC Learning. She found that every class which is learnable using membership queries and equivalence queries is also PAC learnable (Angluin 1988); the PAC learner also works in polynomial time and needs at most polynomially many examples. More recent research on learning Boolean formulas also combines queries with probabilistic aspects (Jackson 1997). Furthermore, query learning has also been applied to ▶ Inductive Inference (see, e.g., Gasarch and Lee 2008; Gasarch and Smith 1992; Jain et al. 2007; Lange and Zilles 2005). Here the power of the learner depends not only on the type of queries permitted but also on whether queries of the corresponding type can be asked finitely often or infinitely often; the latter applies of course only to learning models where the learner converges in the limit and may revise the hypothesis from time to time. Furthermore, queries to oracles have been studied widely; see the entry on ▶ Complexity of Inductive Inference.

Acknowledgements Sanjay Jain was supported in part by NUS grant numbers C252-000-087-001, R146-000-181-112, and R252-000-534-112. Frank Stephen was supported in part by NUS grant numbers R146-000-181-112 and R252-000-534-112.

Recommended Reading

Aizenstein H, Pitt L (1995) On the learnability of disjunctive normal form formulas. Mach Learn 19(3):183–208

Aizenstein H, Hellerstein L, Pitt L (1992) Read-thrice DNF is hard to learn with membership and equivalence queries. In: Thirty-third annual symposium on foundations of computer science, Pittsburgh, 24–27 Oct 1992. IEEE Computer Society, Washington, DC, pp 523–532

Angluin D (1987) Learning regular sets from queries and counterexamples. Info Comput 75(2):87–106

Angluin D (1988) Queries and concept learning. Mach Learn 2(4):319–342

Angluin D (1990) Negative results for equivalence queries. Mach Learn 5:121–150

Angluin D (2004) Queries revisited. Theor Comput Sci 313:175–194

Angluin D, Frazier M, Pitt L (1992) Learning conjunctions of Horn clauses. Mach Learn 9:147–164

Angluin D, Hellerstein L, Karpinski M (1993) Learning read-once formulas with queries. J Assoc Comput Mach 40:185–210

Arias M (2004) Exact learning of first-order Horn expressions from queries. Ph.D. thesis, Tufts University

Arias M, Balcázar JL (2009) Canonical Horn representations and query learning. In: Algorithmic learning theory: twentieth international conference ALT 2009. LNAI, vol 5809. Springer, Berlin, pp 156–170

Arias M, Khardon R (2002) Learning closed Horn expressions. Info Comput 178(1):214–240

Birkendorf A, Böker A, Simon HU (2000) Learning deterministic finite automata from smallest counterexamples. SIAM J Discret Math 13(4):465–491

Hellerstein L, Pillaipakkamnatt K, Raghavan VV, Wilkins D (1996) How many queries are needed to learn? J Assoc Comput Mach 43:840–862

Gasarch W, Lee ACY (2008) Inferring answers to queries. J Comput Syst Sci 74(4):490–512

Gasarch W, Smith CH (1992) Learning via queries. J Assoc Comput Mach 39(3):649–674

Ibarra OH, Jiang T (1988) Learning regular languages from counterexamples. In: Proceedings of the first annual workshop on computational learning theory. MIT, Cambridge/Morgan Kaufmann, San Francisco, pp 371–385

Jackson J (1997) An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. J Comput Syst Sci 55(3):414–440

Jain S, Lange S, Zilles S (2007) A general comparison of language learning from examples and from queries. Theor Comput Sci 387(1):51–66

Lange S, Zilles S (2005) Relations between Gold-style learning and query learning. Infor Comput 203:211–237

Littlestone N (1988) Learning quickly when irrelevant attributes abound: A new linear threshold algorithm. Mach Learn 2:285–318

Maass W, Turán G (1992) Lower bound methods and separation results for on-line learning models. Mach Learn 9:107–145

Radial Basis Function Approximation

▶ Radial Basis Function Networks

Radial Basis Function Networks

Martin D. Buhmann
Justus-Liebig University, Gießen, Germany

Synonyms

Kernel methods; Networks with kernel functions; Neural networks; Quasi-interpolation; Radial basis function approximation; Radial basis function neural networks; Regularization networks; Support vector machines

Definition

Radial basis function networks are a means of approximation by algorithms using linear combinations of translates of a rotationally invariant function, called the radial basis function. The coefficients of these approximations usually solve a minimization problem and can also be computed by interpolation processes. Sometimes the very useful approach of quasi-interpolation is also applied where approximations are computed that do not necessarily match the target functions pointwise but satisfy certain smoothness and decay conditions. The radial basis functions constitute so-called reproducing kernels on certain Hilbert spaces or – in a slightly more general setting – semi-Hilbert spaces. In the latter case, the aforementioned approximation also contains an element from the null-space of the seminorm of the semi-Hilbert space. That is usually a polynomial space.

Motivation and Background

Radial basis function networks are a method to approximate functions and data in a way which is related to the idea of neural networks and learning with kernels. More specifically, approximations of functions or data via algorithms that make use of networks (or neural networks) can be interpreted as either interpolation or minimization problems using kernels of certain shapes, called radial basis functions in the form in which we wish to consider them in this entry. In all cases, they are usually *high-dimensional approximations*, that is, the number of unknowns n in the argument of the kernel may be very large. On the other hand, the number of learning examples ("data") may be quite small. The name neural networks comes from the idea that this learning process simulates the natural functioning of neurons.

At any rate, the purpose of this approach will be the modelization of the learning process by mathematical methods. In most practical cases of

© Springer Science+Business Media New York 2017
C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

networks, the data from which we will learn in the method are rare, i.e., we have few data "points." We will consider this learning approach as an approximation problem in this description; essentially it is a minimizing (regression) problem.

Structure of the Network/Learning System

To begin with, let $\varphi : \mathbb{R}_+ \to \mathbb{R}$ be a univariate continuous function and $\|\cdot\|$ be the Euclidean norm on $\mathbb{R}^n$ for some $n \in \mathbb{N}$, as used for approximation in the seminal paper by Schoenberg (1938). Here, $\mathbb{R}_+$ denotes the set of nonnegative reals. Therefore

$$\varphi(\|\cdot\|) : \mathbb{R}^n \to \mathbb{R},$$
$$(x_1, x_2, \ldots, x_n)^T \mapsto \varphi\left(\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}\right),$$

is a multivariate function and here the number n of unknowns may be very large in practice. This function is rotationally invariant. Incidentally, much of what is going to be said here will work if we replace this function by a general, n-variate function which needs no longer be rotationally invariant, but then, strictly speaking, we are no longer talking about radial basis functions. Then other conditions may replace the restriction to radiality. Nonetheless we stick to the simple case (which is entirely sufficient for many practical applications) when the function really is radially symmetric.

We also require for the time being that this n-variate function be positive definite, that is, for all finite sets Ξ of pairwise different so-called *centers* or data sites $\xi \in \Xi \subset \mathbb{R}^n$, the symmetric matrix

$$A = \{\varphi(\|\xi - \zeta\|)\}_{\xi,\zeta\in\Xi}$$

is a positive definite matrix. The condition of pairwise different data in Ξ may of course in practice not be necessarily met. For quasi-interpolation, no linear systems need to be solved that depend on the target data, but other conditions that guarantee localness and polynomial accuracy of the approximants are required.

This property is usually obtained by requiring that $\varphi(\|\cdot\|)$ be absolutely integrable, and its Fourier transform – which thereby exists and is continuous – is positive everywhere ("Bochner's theorem"). An example for such a useful function is the exponential (the "Gauß-kernel") $\varphi(r) = \exp(-c^2r^2)$, $r \geq 0$, where c is a positive parameter. For this the above positive definiteness is guaranteed for all positive c and all n. Another example is the Poisson-kernel $\varphi(r) = \exp(-cr)$. However, we may also take the non-integrable "inverse multiquadrics" $\varphi(r) = 1/\sqrt{r^2+c^2}$ which has a Fourier transform in the generalized or distributional sense that is also positive everywhere except at zero. There it has a singularity. Nonetheless, the aforementioned matrices of the form A are still always positive definite for these exponentials and the inverse multiquadrics so long as $c > 0$ and $n = 1, 2, \ldots$. Still further examples come from the so-called Dagum class of radial basis functions $\varphi(r) = 1-(r^\beta/(1+r^\beta))^\gamma$ which give positive definiteness for a variety of choices of parameters β and γ.

This requirement of positive definiteness guarantees that for all given finite sets Ξ and "data" $f_\xi \in \mathbb{R}$, $\xi \in \Xi$, there is a unique linear combination:

$$s(x) = \sum_{\xi\in\Xi} \lambda_\xi \varphi(\|x - \xi\|), \qquad x \in \mathbb{R}^n,$$

which satisfies the linear interpolation conditions:

$$s(\xi) = f_\xi, \qquad \forall\ \xi \in \Xi.$$

This is because the interpolation matrix which is used to compute the coefficients λ_ξ is just the matrix A above which is positive definite, thus regular. The expression in the penultimate display is the network that approximates the data given by the user. Of course the interpolation conditions are just what is meant by learning from examples, the data being the $|\Xi|$ examples. Here as always, $|\Xi|$ denotes the cardinality of the set Ξ. In the learning theory, the linear space spanned by the above translates of $\varphi(\|\cdot\|)$ by $\xi \in \Xi$ is called the feature space with φ as activation function.

Incidentally, it is straightforward to generalize the approximation method to an approximation to data in $\mathbb{R}^m$, $m \in \mathbb{N}$, by approximating the data $f_\xi \in \mathbb{R}^m$ componentwise by m such expressions as the above, call them $s_1, s_2, \ldots, s_m$.

Applications

Applications include classification of data, pattern recognition, time series analysis, picture smoothing similar to diffusion methods, and optimization.

Theory/Solution

Returning to interpolation, the problem may also be reinterpreted as a minimization problem. If we define the weighted L^2-integral

$$\|g\|_\varphi := \frac{1}{(2\pi)^{n/2}} \sqrt{\int_{\mathbb{R}^n} \frac{1}{\hat{\varphi}(\|x\|)} |\hat{g}(x)|^2 \, dx}$$

with $\hat{\varphi}$ still being the above positive Fourier transform, for all $g : \mathbb{R}^n \to \mathbb{R}$ for which the Fourier transform in the sense of $L^2(\mathbb{R}^n)$ is well defined and for which the above integral is finite, we may ask for the approximant to the above data – which still must satisfy the aforementioned interpolation conditions – that minimizes $\|\cdot\|_\varphi$. As Duchon noted, for example, for the thin-plate spline case $\varphi(r) = r^2 \log r$ in this seminal paper, this is just the above interpolant, i.e., that linear combination s of translates of radial basis functions, albeit in the thin-plate spline case with a linear polynomial added as we shall see below.

This works immediately both for the two examples of exponential functions and the inverse multiquadrics. Note that the fact that the latter has a Fourier transform with a singularity at the origin does not matter since its reciprocal appears as a weight function in the integral above. The important requirement is that the Fourier transform has no zero. It also works for the positive definite radial basis functions of compact support, for instance, in Buhmann (1998).

Regularization and Generalizations

Since, generally, the interpolation problem to data may be ill conditioned or unsuitable in the face of data errors, smoothing or regularization is appropriate as an alternative. Indeed, the interpolation problem may be replaced by a smoothing problem which is of the form

$$\frac{1}{|\Xi|} \sum_{\xi \in \Xi} \Big(s(\xi) - f_\xi \Big)^2 + \mu \|s\|_\varphi^2 = \min_s !.$$

Here the L^2-integral is still the one used in the description above and μ is a positive smoothing parameter.

However, when there is only a trivial nullspace of the $\|\cdot\|_\varphi$, i.e., $g = 0$ is the only g with $\|g\|_\varphi = 0$, then it is a norm, and the solution of this problem will have the form

$$s(x) = \sum_{\xi \in \Xi} \lambda_\xi \varphi(\|x - \xi\|), \qquad x \in \mathbb{R}^n.$$

This is where the name regularization network comes from, regularization and smoothing being used synonymously. The form used in the penultimate display is a classical regularizing network problem or in the spline terminology a smoothing spline problem. For so-called support vector machines, the square of the residual term $s(\xi) - f_\xi$ should be replaced by another expression, for example, the one by Vapnik (1996):

$$|s(\xi) - f_\xi|_\varepsilon := \begin{cases} f_\xi - s(\xi) - \varepsilon & if\, |f_\xi - s(\xi)| \geq \varepsilon, \\ 0 & \text{otherwise,} \end{cases}$$

and for the support vector machines classification by the truncated power function $(\cdot)_+^\nu$ which is a positive power for positive argument and otherwise zero.

In the case of a classical regularizing network, the coefficients of the solution may be found by solving a similar linear system to the standard interpolation linear system mentioned above, namely,

$$(A + \mu I)\lambda = f,$$

where f is the vector $(f_\xi)_{\xi\in\Xi}$ in $\mathbb{R}^\Xi$ of the data given and $\lambda = (\lambda_\xi)_{\xi\in\Xi}$. The I denotes the $|\Xi| \times |\Xi|$ identity matrix and A is still the same matrix as above. Incidentally, also scaling mechanisms may be introduced into the radial basis function by replacing the simple translate $\varphi(\|x - \xi\|)$ by $\varphi(\|x - \xi\|/\delta)$ for a positive δ which may even depend on ξ.

The ideas of regularization and smoothing are of course not new; for instance, regularization goes back to Tichonov and Arsenin (1977) ("Tichonov regularization") and spline smoothing to Wahba (1985), especially when the smoothing parameter is adjusted via cross-validation or GCV (generalized cross-validation).

Now to the case of semi-norms $\|\cdot\|_\varphi$ with nontrivial null-spaces: indeed, the same idea can be carried through for other radial basis functions as well. In particular we are thinking here of those ones that do not provide positive definite radial basis interpolation matrices but strictly conditionally positive definite ones. We have strictly positive definite radial basis functions of order $k + 1$, $k \geq -1$, if the above interpolation matrices A are still positive definite but only on the subspace of those nonzero vectors $\lambda = (\lambda_\xi)$ in $\mathbb{R}^\Xi$ which satisfy

$$\sum_{\xi\in\Xi} \lambda_\xi p(\xi) = 0 \qquad \forall\ p \in \mathbb{P}_n^k,$$

where $\mathbb{P}_n^k$ denotes the linear space of polynomials in n variables with total degree at most k. In other words, the quadratic form $\lambda^T A\lambda$ need only be positive for such $\lambda \neq 0$. For simplicity of the presentation, we shall let $\mathbb{P}_n^{-1}$ denote $\{0\}$. In particular, if the radial basis function is conditionally positive definite of order 0, its interpolation matrices A are always positive definite, that is, without condition. Also, we have the minimal requirement that the sets of centers Ξ are unisolvent for this polynomial space, i.e., the only polynomial $p \in \mathbb{P}_n^k$ that vanishes identically on Ξ is the zero polynomial.

The connection of this with a layered neural network is that the approximation above is a weighted sum (weighted by the coefficients λ_ξ) over usually nonlinear activation functions φ. The entries in the sum are the radial basis function neurons and there are usually many of them. The number of nodes in the model is n. The hidden layer of "radial basis function units" consists of $|\Xi|$ nodes, i.e., the number of centers in our radial basis function approximation. The output layer has m responses if the radial basis function approximation above is generalized to m-variate data, i.e., then we get $s_1, s_2, \ldots, s_m$ instead of just s, as already described. This network here is of the type of a nonlinear, layered, feedforward network. More than one hidden layer is unusual. The choice of the radial basis functions (its smoothness, for instance) and the flexibility in the positioning of the centers in clusters, grids (e.g., Buhmann 1990), or otherwise provide much of the required freedom for good approximations.

The properties of conditional positive definiteness are fulfilled now for a much larger realm of radial basis functions which have still nowhere vanishing generalized Fourier transforms but with higher-order singularities at the origin. (Remember that this creates no problem for the well definedness of $\|\cdot\|_\varphi$.) For instance, the above properties are true for the thin-plate spline function $\varphi(r) = r^2 \log r$, for the shifted logarithm $\varphi(r) = (r^2 + c^2)\log(r^2 + c^2)$, and for the multiquadric $\varphi(r) = -\sqrt{r^2 + c^2}$. Here we still have a parameter c which may now be arbitrary real. The order of the above is one for the multiquadric and two for the thin-plate spline. Another commonly used radial basis function which gives rise to conditional positive definiteness is the $\varphi(r) = r^3$.

Hence the norm becomes a semi-norm with null-space $\mathbb{P}_n^k$, but it still has the same form as a square integral with the reciprocal of the Fourier transform of the radial basis function as a weight.

Therefore we have to include a polynomial from the null-space of the semi-norm to the approximant which becomes

$$s(x) = \sum_{\xi \in \Xi} \lambda_\xi \varphi(\|x - \xi\|) + q(x), \qquad x \in \mathbb{R}^n,$$

where $q \in \mathbb{P}_n^k$ and the side conditions on the coefficients

$$\sum_{\xi \in \Xi} \lambda_\xi p(\xi) = 0, \qquad \forall\ p \in \mathbb{P}_n^k.$$

When quasi-interpolation is used, this inclusion of polynomials is not because they are not formed by interpolation condition, and the reproduction of polynomials (thus their presence in the linear space) is directly guaranteed by their construction.

If we consider the regularization network problem with the smoothing parameter μ again, then we have to solve the linear system with a smoothing parameter μ:

$$(A + \mu I)\lambda + P^T b = f, \quad P\lambda = 0,$$

where $P = (p_i(\xi))_{i=1,\ldots,L,\xi \in \Xi}$ and p_i form a basis of $\mathbb{P}_n^k$, b_i being the components of b, and $q(x) = \sum_{i=1}^{L} b_i p_i(x)$ is the expression of the polynomial added to the radial basis function sum. So in particular P is a matrix with as many rows as the dimension $L = \binom{n+k}{n}$ of $\mathbb{P}_n^k$ is and $|\Xi|$ columns.

In all cases, the radial basis functions composed of the Euclidean norm can be regarded as reproducing kernels in the semi-Hilbert spaces defined by the set X of distributions f for which $\|g\|_\varphi$ is finite and the semi-inner product

$$(h, g) = \frac{1}{(2\pi)^n} \int_{\mathbb{R}^n} \frac{1}{\hat{\varphi}(\|x\|)} \hat{h}(x)\overline{\hat{g}(x)}\, dx,$$
$$h, g, \in X.$$

In particular, $\|g\|_\varphi^2 = (g, g)$. If the evaluation functional is continuous (bounded) on that space X, there exists a reproducing kernel, i.e., there is a $K : X \times X \to \mathbb{R}$ such that

$$g(x) = (g, K(\cdot, x)), \quad \forall\ x \in \mathbb{R}^n,\ g \in X,$$

See, for example, Wahba (1990). If the semi-inner product is actually an inner product, then the reproducing kernel is unique. The kernel gives rise to positive definite matrices $\{K(\xi, \zeta)\}_{\xi,\zeta \in \Xi}$ if and only if it is a positive operator. For the spaces X defined by our radial basis functions, it turns out that $K(x, y) := \varphi(\|x - y\|)$; see, e.g., the overview in Buhmann (2003). Then the matrices A are positive definite if $\hat{\varphi}(\|\cdot\|)$ is well defined and positive, but if it has a singularity at zero, the A may be only conditionally positive definite. Note here that $\hat{\varphi}(\|\cdot\|)$ denotes the n-variate Fourier transform of $\varphi(\|\cdot\|)$, both being radially symmetric.

Advantages of the Approach

Why are we interested in using radial basis functions for networks? The radial basis functions have many excellent approximation properties which make them useful as general tools for approximation. Among them are the variety of more or less smoothness as required (e.g., multiquadrics is C^∞ for positive c and just continuous for $c = 0$), the fast evaluation and computation methods available (see, e.g., Beatson and Powell 1994), the aforementioned nonsingularity properties and their connection with the theory of reproducing kernel Hilbert spaces, and finally their excellent convergence properties (see, e.g., Buhmann 2003). Generally, neural networks are a tried and tested approach to approximation, modeling, and smoothing by methods from learning theory.

Limitations

The number of applications where the radial basis function approach has been used is vast. Also, the solutions may be computed efficiently by far-field expansions, approximated Lagrange functions, and multipole methods. However, there are still some limitations with these important computational methods when the dimension n is large. So far, most of the multipole and far-field methods have been implemented only for medium-sized dimensions.

Cross-References

▶ Neural Networks
▶ Regularization

Recommended Reading

Beatson RK, Powell MJD (1994) An iterative method for thin plate spline interpolation that employs approximations to Lagrange functions. In: Griffiths DF, Watson GA (eds) Numerical analysis 1993. Longman, Burnt Mill, pp 17–39

Broomhead D, Lowe D (1988) Radial basis functions, multi-variable functional interpolation and adaptive networks. Complex Syst 2:321–355

Buhmann MD (1990) Multivariate cardinal-interpolation with radial-basis functions. Construct Approx 6:225–255

Buhmann MD (1993) On quasi-interpolation with radial basis functions. J Approx Theory 72:103–130

Buhmann MD (1998) Radial functions on compact support. Proc Edinb Math Soc 41:33–46

Buhmann MD (2003) Radial basis functions: theory and implementations. Cambridge University Press, Cambridge

Buhmann MD, Porcu E, Daley D, Bevilacqua M (2013) Radial basis functions for multivariate geostatistics. Stoch Env Res Risk Assess 27(4): 909–922

Duchon J (1976) Interpolation des fonctions de deux variables suivant le principe de la flexion des plaques minces. RAIRO 10:5–12

Evgeniou T, Poggio T, Pontil M (2000) Regularization networks and support vector machines. Adv Comput Math 13:1–50

Hardy RL (1990) Theory and applications of the multiquadric-biharmonic method. Comput Math Appl 19:163–208

Micchelli CA (1986) Interpolation of scattered data: distance matrices and conditionally positive definite functions. Construct Approx 1:11–22

Pinkus A (1996) TDI-subpaces of $C(\mathbb{R}^d)$ and some density problems from neural networks. J Approx Theory 85:269–287

Schoenberg IJ (1938) Metric spaces and completely monotone functions. Ann Math 39:811–841

Tichonov AN, Arsenin VY (1977) Solution of Ill-posed problems. W.H. Winston, Washington, DC

Vapnik VN (1996) Statistical learning theory. Wiley, New York

Wahba G (1985) A comparison of GCV and GML for choosing the smoothing parameter in the generalized splines smoothing problem. Ann Stat 13:1378–1402

Wahba G (1990) Spline models for observational data. Series in applied mathematics, vol 59. SIAM, Philadelphia

Radial Basis Function Neural Networks

▶ Radial Basis Function Networks

Random Decision Forests

▶ Random Forests

Random Forests

Synonyms

Random decision forests

Definition

Random Forests is an ▶ ensemble learning technique. It is a hybrid of the ▶ Bagging algorithm and the ▶ random subspace method, and uses ▶ decision trees as the base classifier. Each tree is constructed from a bootstrap sample from the original dataset. An important point is that the trees are not subjected to pruning after construction, enabling them to be partially overfitted to their own sample of the data. To further diversify the classifiers, at each branch in the tree, the decision of which feature to split on is restricted to a *random subset* of size n, from the full feature set. The random subset is chosen anew for each branching point. n is suggested to be $\log_2(N+1)$, where N is the size of the whole feature set.

Random Subspace Method

Synonyms

Random subspaces; RSM

Definition

The random subspace method is an ▶ ensemble learning technique. The principle is to increase diversity between members of the ensemble by restricting classifiers to work on different random subsets of the full feature space. Each classifier learns with a subset of size n, chosen uniformly at random from the full set of size N. Empirical studies have suggested good results can be obtained with the rule-of-thumb to choose $n = N/2$ features. The method is generally found to perform best when there are a large number of features (large N), and the discriminative information is spread across them. The method can underperform in the converse situation, when there are few informative features, and a large number of noisy/irrelevant features. ▶ Random Forests is an algorithm combining RSM with the ▶ Bagging algorithm, which can provide significant gains over each used separately.

Random Subspaces

▶ Random Subspace Method

Randomized Decision Rule

▶ Markovian Decision Rule

Randomized Experiments

▶ Online Controlled Experiments and A/B Testing

Rank Correlation

Johannes Fürnkranz[1,2] and Eyke Hüllermeier[3]
[1] Knowledge Engineering Group, TU Darmstadt, Darmstadt, Deutschland
[2] Department of Information Technology, University of Leoben, Leoben, Austria
[3] Department of Computer Science, Paderborn University, Paderborn, Germany

Abstract

Rank correlation measures the correspondence between two rankings, τ and τ', of a set of m objects.

Method

Various proposals for such measures have been made, especially in the field of statistics. Two of the best-known measures are Spearman's rank correlation and Kendall's tau.

Spearman's rank correlation (Spearman 1904) calculates the sum of squared rank distances and is normalized such that it evaluates to -1 for reversed and to $+1$ for identical rankings. Formally, it is defined as follows:

$$(\tau, \tau') \mapsto 1 - \frac{6 \sum_{i=1}^{m} (\tau(i) - \tau'(i))^2}{m(m^2 - 1)} \qquad (1)$$

Kendall's tau (Kendall 1938) is the number of pairwise rank inversions between τ and τ', again normalized to the range $[-1, +1]$:

$$(\tau, \tau') \mapsto 1 - \frac{4 \left| \{(i, j) \mid i < j, \tau(i) < \tau(j) \wedge \tau'(i) > \tau'(j)\} \right|}{m(m-1)} \qquad (2)$$

Spearman's rank correlation and Kendall's tau give equal weight to all ranking positions, which is not desirable for all applications. For example, ranking problems in information retrieval are often evaluated with the (normalized) discounted cumulative gain (NDCG), which assigns more weight to the lower-ranking positions (cf. ▶ learning to rank).

Cross-References

▶ Learning to Rank
▶ Preference Learning
▶ ROC Analysis

Recommended Reading

Kendall M (1938) A new measure of rank correlation. Biometrika 30(1):81–89

Spearman C (1904) The proof and measurement of association between two things. Am J Psychol 15: 2–101

Ratio Scale

A **ratio** measurement scale possesses all the characteristics of interval measurement, and there exists a *zero* that, the same as arithmetic *zero*, means "nil" or "nothing." See ▶ Measurement Scales.

Real-Time Dynamic Programming

Real-Time Dynamic Programming (RTDP) is the same as ▶ Adaptive Real-Time Dynamic Programming (ARTDP) without the system identification component. It is applicable when an accurate model of the problem is available. It converges to an optimal policy of a stochastic optimal path problem under suitable conditions. RTDP was introduced by Barto et al. (1995) in their paper Learning to Act Using RTDP.

Recall

Recall is a measure of information retrieval performance. Recall is the total number of documents retrieved that are elevant/Total number of relevant documents in the database. See ▶ Precision and Recall.

Cross-References

▶ Sensitivity

Receiver Operating Characteristic Analysis

▶ ROC Analysis

Recognition

▶ Classification

Recommender Systems

Prem Melville and Vikas Sindhwani
IBM T. J. Watson Research Center, Yorktown Heights, NY, USA

Definition

The goal of a recommender system is to generate meaningful recommendations to a collection of users for items or products that might interest them. Suggestions for books on Amazon, or movies on Netflix, are real-world examples of the operation of industry-strength recommender systems. The design of such recommendation engines depends on the domain and the particular characteristics of the data available. For example, movie watchers on Netflix frequently provide ratings on a scale of 1 (disliked) to 5 (liked). Such a data source records the quality of interactions between users and items. Additionally, the system may have access to user-specific and item-specific profile attributes such as demographics and product descriptions, respectively. Recommender systems differ in the way they analyze these data sources to develop notions of affinity between users and items, which can be used to identify well-matched pairs. ► Collaborative Filtering systems analyze historical interactions alone, while ► Content-based Filtering systems are based on profile attributes; and hybrid techniques attempt to combine both of these designs. The architecture of recommender systems and their evaluation on real-world problems is an active area of research.

Motivation and Background

Obtaining recommendations from trusted sources is a critical component of the natural process of human decision making. With burgeoning consumerism buo-yed by the emergence of the web, buyers are being presented with an increasing range of choices while sellers are being faced with the challenge of personalizing their advertising efforts. In parallel, it has become common for enterprises to collect large volumes of transactional data that allows for deeper analysis of how a customer base interacts with the space of product offerings. Recommender systems have evolved to fulfill the natural dual need of buyers and sellers by automating the generation of recommendations based on data analysis.

The term "collaborative filtering" was introduced in the context of the first commercial recommender system, called Tapestry (Goldberg et al. 1992), which was designed to recommend documents drawn from newsgroups to a collection of users. The motivation was to leverage social collaboration in order to prevent users from getting inundated by a large volume of streaming documents. Collaborative filtering, which analyzes usage data across users to find well-matched user-item pairs, has since been juxtaposed against the older methodology of content filtering, which had its original roots in information retrieval. In content filtering, recommendations are not "collaborative" in the sense that suggestions made to a user do not explicitly utilize information across the entire user-base. Some early successes of collaborative filtering on related domains included the GroupLens system (Resnick et al. 1994b).

As noted in Billsus and Pazzani (1998), initial formulations for recommender systems were based on straightforward correlation statistics and predictive modeling, not engaging the wider range of practices in statistics and machine learning literature. The collaborative filtering problem was mapped to classification, which allowed dimensionality reduction techniques to be brought into play to improve the quality of the solutions. Concurrently, several efforts attempted to combine content-based methods with collaborative filtering, and to incorporate additional domain knowledge in the architecture of recommender systems.

Further research was spurred by the public availability of datasets on the web, and the interest generated due to direct relevance to e-commerce. Netflix, an online streaming video and DVD rental service, released a large-scale dataset containing 100 million ratings given by about half-a-million users to thousands of movie titles, and announced an open competition for the best collaborative filtering algorithm in this domain. Matrix Factorization (Bell et al. 2009) techniques rooted in numerical linear algebra and statistical matrix analysis emerged as a state-of-the-art technique.

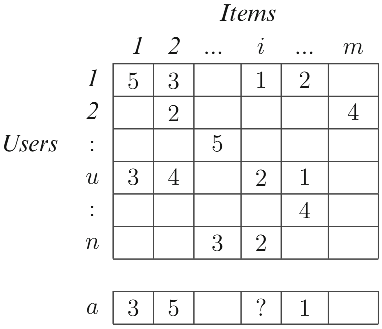
Currently, recommender systems remain an active area of research, with a dedicated ACM conference, intersecting several subdisciplines of statistics, machine learning, data mining, and information retrievals. App-lications have been pursued in diverse domains ranging from recommending webpages to music, books, movies, and other consumer products.

Structure of Learning System

The most general setting in which recommender systems are studied is presented in Fig. 1. Known user preferences are represented as a matrix of n users and m items, where each cell $r_{u,i}$ corresponds to the rating given to item i by the user u. This *user ratings matrix* is typically sparse, as most users do not rate most items. The *recommendation task* is to predict what rating a user would give to a previously unrated item. Typically, ratings are predicted for all items that have not been observed by a user, and the highest rated items are presented as recommendations. The user under current consideration for recommendations is referred to as the *active user*.

The myriad approaches to recommender systems can be broadly categorized as:

- *Collaborative Filtering (CF)*: In CF systems, a user is recommended items based on the past ratings of all users collectively.
- *Content-based recommending*: These approaches recommend items that are similar in content to items the user has liked in the past, or matched to pre-defined attributes of the user.
- *Hybrid approaches*: These methods combine both collaborative and content-based approaches.



		Items					
		1	*2*	...	*i*	...	*m*
Users	*1*	5	3		1	2	
	2		2				4
	:			5			
	u	3	4		2	1	
	:					4	
	n			3	2		
	a	3	5		?	1	

Recommender Systems, Fig. 1 User ratings matrix, where each cell $r_{u,i}$ corresponds to the rating of user u for item i. The task is to predict the missing rating $r_{a,i}$ for the active user a

Collaborative Filtering

Collaborative filtering (CF) systems work by collecting user feedback in the form of ratings for items in a given domain and exploiting similarities in rat-ing behavior amongst several users in determining how to recommend an item. CF methods can be further subdivided into *neighborhood-based* and *model-based* approaches. Neighborhood-based methods are also commonly referred to as *memory-based* approaches (Breese et al. 1998).

Neighborhood-Based Collaborative Filtering

In neighborhood-based techniques, a subset of users are chosen based on their similarity to the active user, and a weighted combination of their ratings is used to produce predictions for this user. Most of these approaches can be generalized by the algorithm summarized in the following steps:

1. Assign a weight to all users with respect to similarity with the active user.
2. Select k users that have the highest similarity with the active user – commonly called the *neighborhood*.
3. Compute a prediction from a weighted combination of the selected neighbors' ratings.

In step 1, the weight $w_{a,u}$ is a measure of similarity between the user u and the active user a. The most commonly used measure of similarity is the Pearson correlation coefficient between the ratings of the two users (Resnick et al. 1994a), defined below:

$$w_{a,u} = \frac{\sum_{i \in I}(r_{a,i} - \bar{r}_a)(r_{u,i} - \bar{r}_u)}{\sqrt{\sum_{i \in I}(r_{a,i} - \bar{r}_a)^2 \sum_{i \in I}(r_{u,i} - \bar{r}_u)^2}} \qquad (1)$$

where I is the set of items rated by both users, $r_{u,i}$ is the rating given to item i by user u, and $\bar{r}_u$ is the mean rating given by user u.

In step 3, predictions are generally computed as the weighted average of deviations from the neighbor's mean, as in:

$$p_{a,i} = \bar{r}_a + \frac{\sum_{u \in K}(r_{u,i} - \bar{r}_u) \times w_{a,u}}{\sum_{u \in K} w_{a,u}} \qquad (2)$$

where $p_{a,i}$ is the prediction for the active user a for item i, $w_{a,u}$ is the similarity between users a and u, and K is the neighborhood or set of most similar users.

Similarity based on Pearson correlation measures the extent to which there is a linear dependence between two variables. Alternatively, one can treat the ratings of two users as a vector in an m-dimensional space, and compute similarity based on the cosine of the angle between them, given by:

$$W_{a,u} = \cos(\mathbf{r}_a, \mathbf{r}_u) = \frac{\mathbf{r}_a \cdot \mathbf{r}_u}{\|\mathbf{r}_a\|_2 \times \|r_u\|_2} \\ = \frac{\sum_{i=1}^{m} r_{a,i} r_{u,i}}{\sqrt{\sum_{i=1}^{m} r_{a,i}^2}\sqrt{\sum_{i=1}^{m} r_{u,i}^2}} \qquad (3)$$

When computing cosine similarity, one cannot have negative ratings, and unrated items are treated as having a rating of zero. Empirical studies (Breese et al. 1998) have found that Pearson correlation generally performs better. There have been several other similarity measures used in the literature, including *Spearman rank correlation*, *Kendall's* τ *correlation*, *mean squared differences*, *entropy*, and *adjusted cosine similarity* (Herlocker et al. 1999; Su and Khoshgoftaar 2009).

Several extensions to neighborhood-based CF, which have led to improved performance are discussed below.

Item-based Collaborative Filtering: When applied to millions of users and items, conventional neighborhood-based CF algorithms do not scale well, because of the computational complexity of the search for similar users. As a alternative, Linden et al. (2003) proposed *item-to-item* collaborative filtering where rather than matching similar users, they match a user's rated items to similar items. In practice, this approach leads to faster online systems, and often results in improved recommendations (Linden et al. 2003; Sarwar et al. 2001).

In this approach, similarities between pairs of items i and j are computed off-line using Pearson correlation, given by:

$$w_{i,j} = \frac{\sum_{u \in U}(r_{u,i} - \bar{r}_i)(r_{u,j} - \bar{r}_j)}{\sqrt{\sum_{u \in U}(r_{u,i} - \bar{r}_i)^2}\sqrt{\sum_{u \in U}(r_{u,j} - \bar{r}_j)^2}} \qquad (4)$$

where U is the set of all users who have rated both items i and j, $r_{u,i}$ is the rating of user u on item i, and $j, r_{u,i}$ is the average rating of the ith item across users.

Now, the rating for item i for user a can be predicted using a simple weighted average, as in:

$$p_{a,i} = \frac{\sum_{j \in K} r_{a,j} w_{i,j}}{\sum_{j \in K} |w_{i,j}|} \qquad (5)$$

where K is the neighborhood set of the k items rated by a that are most similar to i.

For item-based collaborative filtering too, one may use alternative similarity metrics such as *adjusted cosine similarity*. A good empirical comparison of variations of item-based methods can be found in Sarwar et al. (2001).

Significance Weighting: It is common for the active user to have highly correlated neighbors that are based on very few co-rated (overlapping) items. These neighbors based on a small number of overlapping items tend to be bad predictors. One approach to tackle this problem is to multiply the similarity weight by a *significance weighting* factor, which devalues the correlations based on few co-rated items (Herlocker et al. 1999).

Default Voting: An alternative approach to dealing with correlations based on very few co-rated items is to assume a default value for the rating for items that have not been explicitly rated. In this way one can now compute correlation (Eq. 1) using the union of items rated by users being matched as opposed to the intersection. Such a *default voting* strategy has been shown to improve collaborative filtering by Breese et al. (1998).

Inverse User Frequency: When measuring the similarity between users, items that have been rated by all (and universally liked or disliked) are not as useful as less common items. To account for this Breese et al. (1998) introduced the notion of *inverse user frequency*, which is computed as $f_i = \log n/n_i$, where n_i is the number of users who have rated item i out of the total number of n users. To apply inverse user frequency while using similarity-based CF, the original rating is transformed for i by multiplying it by the factor f_i. The underlying assumption of this approach is that items that are universally loved or hated are rated more frequently than others.

Case Amplification: In order to favor users with high similarity to the active user, Breese et al. (1998) introduced *case amplification* which transforms the original weights in Eq. (2) to

$$w'_{a,u} = w_{a,u} \cdot |w_{a,u}|^{\rho-1}$$

where ρ is the amplification factor, and $\rho \geq 1$.

Other notable extensions to similarity-based collaborative filtering include *weighted majority prediction* (Nakamura and Abe 1998) and *imputation-boosted CF* (Su et al. 2008).

Model-based Collaborative Filtering: Model-based techniques provide recommendations by estimating parameters of statistical models for user ratings. For example, Billsus and Pazzani (1998) describe an early approach to map CF to a classification problem, and build a classifier for each active user representing items as features over users and available ratings as labels, possibly in conjunction with dimensionality reduction techniques to overcome data sparsity issues. Other predictive modeling techniques have also been applied in closely related ways.

More recently, ▶ latent factor and matrix factorization models have emerged as a state-of-the-art methodology in this class of techniques (Bell et al. 2009). Unlike neighborhood based methods that generate recommendations based on statistical notions of similarity between users, or between items, latent factor models assume that the similarity between users and items is simultaneously induced by some hidden lower-dimensional structure in the data. For example, the rating that a user gives to a movie might be assumed to depend on few implicit factors such as the user's taste across various movie genres. Matrix factorization techniques are a class of widely successful latent factor models where users and items are simultaneously represented as unknown feature vectors (column vectors) w_u, $h_i \in \mathbb{R}^k$ along k latent dimensions. These feature vectors are learnt so that inner products $w_u^T h_i$ approximate the known preference ratings $r_{u,i}$ with respect to some loss measure. The squared loss is a standard choice for the loss function, in which case the following objective function is minimized,

$$j(W, H) = \sum_{(u,i)\in L} (r_{u,i} - w_u^T h_i)^2 \qquad (6)$$

where $W = [w_1 \ldots w_n]^T$ is an $n \times k$ matrix, $H = [h_1 \ldots h_m]$ is a $k \times m$ matrix, and L is the set of user-item pairs for which the ratings are known. In the impractical limit where all user-item ratings are known, the above objective function is $J(W, H) = \|R - WH\|^2_{fro}$ where R denotes the $n \times m$ fully known user-item matrix. The solution to this problem is given by taking the truncated SVD of R, $R = UDV^T$ and setting $W = U_K D_k^{\frac{1}{2}}$, $H = D_k^{\frac{1}{2}} V_k^T$ where U_k, D_k, V_k contain the k largest singular triplets of R. However, in the realistic setting where the majority of user-item ratings are unknown and insufficient number of matrix entries are observed, such a nice globally optimal solution cannot in general be directly obtained, and one has to explicitly optimize the non-convex objective function $J(W, H)$. Note that in this case, the objective function is a particular form of weighted loss, that is, $J(W, H) = \|S \odot (R - WH)\|^2_{fro}$ where $\odot$ denotes elementwise products, and S is a binary matrix that equals one over known user-item pairs L, and 0 otherwise. Therefore, weighted low-rank approximations are pertinent to this discussion (Srebro and Jaakkola 2003). Standard optimization procedures include gradient-based techniques, or procedures like alternating least squares where H is solved keeping W fixed and vice versa until a convergence criterion is satisfied. Note that fixing either W or H turns the problem of estimating the other into a weighted ▶ linear regression task. In order to avoid learning a model that overfits, it is common to minimize the objective function in the presence of ▸ regularization terms, $J(W, H) + \gamma\|W\|^2 + \lambda\|H\|^2$, where γ, λ are regularization parameters that can be determined by cross-validation. Once W, H are learnt, the product WH provides an approximate reconstruction of the rating matrix from where recommendations can be directly read off.

Different choices of loss functions, regularizers, and additional model constraints have generated a large body of literature on matrix factorization techniques. Arguably, for discrete ratings, the squared loss is not the most natural loss function. The maximum margin matrix factorization (Rennie and Srebro 2005) approach uses margin-based loss functions such as the hinge loss used in ▸ SVM classification, and its ordinal extensions for handling multiple ordered rating categories. For ratings that span over K values, this reduces to finding $K - 1$ thresholds that divide the real line into consecutive intervals specifying rating bins to which the output is mapped, with a penalty for insufficient margin of separation. Rennie and Srebro (2005) suggest a nonlinear conjugate gradient algorithm to minimize a smoothed version of this objective function.

Another class of techniques is the nonnegative matrix factorization popularized by the work of Lee and Seung (1999) where nonnegativity constraints are imposed on W, H. There are weighted extensions of NMF that can be applied to recommendation problems. The rating behavior of each user may be viewed as being a manifestation of different roles, for example, a composition of prototypical behavior in clusters of users bound by interests or community. Thus, the ratings of each user are an additive sum of basis vectors of ratings in the item space. By disallowing subtractive basis, nonnegativity constraints lend a part-based interpretation to the model. NMF can be solved with a variety of loss functions, but with the generalized KL-divergence loss defined as follows,

$$J(W, H) = \sum_{u,i \in L} r_{u,i} \log \frac{r_{u,i}}{w_u^T h_i} - r_{u,i} + w_u^T h_i$$

NMF is in fact essentially equivalent to probabilistic latent semantic analysis (pLSA) which has also previously been used for collaborative filtering tasks (Hofmann 2004).

The recently concluded million-dollar Netflix competition has catapulted matrix factorization techniques to the forefront of recommender technologies in collaborative filtering settings (Bell et al. 2009). While the final winning solution was a complex ensemble of different models, several enhancements to basic matrix factorization models were found to lead to improvements. These included:

1. The use of additional user-specific and item-specific parameters to account for systematic biases in the ratings such as popular movies receiving higher ratings on average.
2. Incorporating temporal dynamics of rating behavior by introducing time-dependent variables.

In many settings, only implicit preferences are available, as opposed to explicit like–dislike ratings. For example, large business organizations, typically, meticulously record transactional details of products purchased by their clients. This is a one-class setting since the business domain knowledge for negative examples that a client has no interest in buying a product ever in the future is typically not available explicitly in corporate databases. Moreover, such knowledge is difficult to gather and maintain in the first place, given the rapidly changing business environment. Another example is recommending TV shows based on watching habits of users, where preferences are implicit in what the users chose to see without any source of explicit ratings. Recently, matrix factorization techniques have been advanced to handle such problems (Pan and Scholz 2009) by formulating confidence weighted objective function, $J(W, H) = \Sigma_{(u,i)} c_{u,i}(r_{u,i} - w_u^T h_i)^2$, under the assumption that unobserved user-item pairs may be taken as negative examples with a certain degree of confidence specified via $c_{u,i}$.

The problem of recovering missing values in a matrix from a small fraction of observed entries is also known as the Matrix Completion problem. Recent work by Candès and Tao (2009) and Recht (2009) has shown that under certain assumptions on the singular vectors of the matrix, the matrix completion problem can be solved exactly by a convex optimization problem provided with a sufficient number of observed entries. This problem involves finding among all matrices consistent with the observed entries, the one with the minimum nuclear norm (sum of singular values).

Content-Based Recommending

Pure collaborative filtering recommenders only utilize the user ratings matrix, either directly, or to induce a collaborative model. These approaches treat all users and items as atomic units, where predictions are made without regard to the specifics of individual users or items. However, one can make a better personalized recommendation by knowing more about a user, such as demographic information (Pazzani 1999), or about an item, such as the director and genre of a movie (Melville et al. 2002). For instance, given movie genre information, and knowing that a user liked "Star Wars" and "Blade Runner," one may infer a predilection for science fiction and could hence recommend "Twelve Monkeys." Content-based recommenders refer to such approaches, that provide recommendations by comparing representations of content describing an item to representations of content that interests the user. These approaches are sometimes also referred to as *content-based filtering*.

Much research in this area has focused on recommending items with associated *textual* content, such as web pages, books, and movies; where the web pages themselves or associated content like descriptions and user reviews are available. As such, several approaches have treated this problem as an information retrieval (IR) task, where the content associated with the user's preferences is treated as a query, and the unrated documents are scored with relevance/similarity to this query (Balabanovic and Shoham 1997). In NewsWeeder (Lang 1995), documents in each rating category are converted into *tf-idf* word vectors, and then averaged to get a prototype vector of each category for a user. To classify a new document, it is compared with each prototype vector and given a predicted rating based on the cosine similarity to each category.

An alternative to IR approaches, is to treat recommending as a classification task, where each example represents the content of an item, and a user's past ratings are used as labels for these examples. In the domain of book recommending, Mooney and Roy (2000) use text from fields such as the title, author, synopses, reviews, and subject terms, to train a multinomial naive Bayes classifier. Ratings on a scale of 1 to k can be directly mapped to k classes (Melville et al. 2002), or alternatively, the numeric rating can be used to weight the training example in a probabilistic binary classification setting (Mooney and Roy 2000). Other classification algorithms have also been used for purely content-based recommending, including k-nearest neighbor, ▶ decision trees, and ▶ neural networks (Pazzani and Billsus 1997).

Hybrid Approaches

In order to leverage the strengths of content-based and collaborative recommenders, there have been several hybrid approaches proposed that combine the two. One simple approach is to allow both content-based and collaborative filtering methods to produce separate ranked lists of recommendations, and then merge their results to produce a final list (Cotter and Smyth 2000). Claypool et al. (1999) combine the two predictions using an adaptive weighted average, where the weight of the collaborative component increases as the number of users accessing an item increases.

Melville et al. (2002) proposed a general framework for *content-boosted collaborative filtering*, where content-based predictions are applied to convert a sparse user ratings matrix into a full ratings matrix, and then a CF method is used to provide recommendations. In particular, they use a Naïve Bayes classifier trained on documents describing the rated items of each user, and replace the unrated items by predictions from this classifier. They use the resulting *pseudo ratings matrix* to find neighbors similar to the active user, and produce predictions using Pearson correlation, appropriately weighted to account for the overlap of actually rated items, and for the active user's content predictions. This approach has been shown to perform better than pure collaborative filtering, pure content-based systems, and a linear combination of the two. Within this content-boosted CF framework, Su et al. (2007) demonstrated improved results using a stronger content-predictor, TAN-ELR, and unweighted Pearson collaborative filtering.

Several other hybrid approaches are based on traditional collaborative filtering, but also maintain a content-based profile for each user. These content-based profiles, rather than co-rated items, are used to find similar users. In Pazzani's approach (Pazzani 1999), each user-profile is represented by a vector of weighted words derived from positive training examples using the Winnow algorithm. Predictions are made by applying CF directly to the matrix of user-profiles (as opposed to the user-ratings matrix). An alternative approach, Fab (Balabanovic and Shoham 1997), uses ▶ relevance feedback to simultaneously mold a personal filter along with a communal "topic" filter. Documents are initially ranked by the topic filter and then sent to a user's personal filter. The user's relevance feedback is used to modify both the personal filter and the originating topic filter. Good et al. (1999) use collaborative filtering along with a number of personalized information filtering agents. Predictions for a user are made by applying CF on the set of other users and the active user's personalized agents.

Several hybrid approaches treat recommending as a classification task, and incorporate collaborative elements in this task. Basu et al. (1998) use *Ripper*, a ▶ rule induction system, to learn a function that takes a user and movie and predicts whether the movie will be liked or disliked. They combine collaborative and content information, by creating features such as *comedies liked by user* and *users who liked movies of genre X*. In other work, Soboroff and Nicholas (1999) multiply a *term-document matrix* representing all item content with the user-ratings matrix to produce a *content-profile matrix*. Using latent semantic Indexing, a rank-k approximation of the content-profile matrix is computed. Term vectors of the user's relevant documents are averaged to produce a user's profile. Then, new documents are ranked against each user's profile in the LSI space. Some hybrid approaches attempt to directly combine content and collaborative data under a single probabilistic framework. Popescul et al. (2001) extended Hofmann's *aspect model* (Hofmann 1999) to incorporate a three-way co-occurrence data among users, items, and item content. Their generative model assumes that users select latent topics, and documents and their content words are generated from these topics. Schein et al. (2002) extend this approach, and focus on making recommendations for items that have not been rated by any user.

Evaluation Metrics

The quality of a recommender system can be evaluated by comparing recommendations to a test set of known user ratings. These systems are typical measured using *predictive accuracy metrics* (Herlocker et al. 2004), where the predicted ratings are directly compared to actual user ratings. The most commonly used metric in the literature is ▶ Mean Absolute Error (MAE) – defined as the average absolute difference between predicted ratings and actual ratings, given by:

$$\text{MAE} = \frac{\sum_{\{u,i\}} |P_{u,i} - r_{u,i}|}{N} \tag{7}$$

Where $p_{u,i}$ is the predicted rating for user u on item i, $r_{u,i}$ is the actual rating, and N is the total number of ratings in the test set.

A related commonly used metric, Root Mean Squared Error (RMSE), puts more emphasis on larger absolute errors, and is given by:

$$\text{RMSE} = \sqrt{\frac{\sum_{\{u,i\}} (p_{u,i} - r_{u,i})^2}{N}} \tag{8}$$

Predictive accuracy metrics treat all items equally. However, for most recommender systems the primary concern is accurately predict the items a user will like. As such, researchers often view recommending as predicting *good*, that is, items with high ratings versus *bad* or poorly rated items. In the context of information retrieval (IR), identifying the good from the background of bad items can be viewed as discriminating between "relevant" and "irrelevant" items; and as such, standard IR measures, like ▶ Precision, ▶ Recall and ▶ Area Under the ROC Curve (AUC) can be utilized. These, and several other measures, such as *F1-measure*, *Pearson's product-moment correlation*, *Kendall's* τ, *mean average precision*, *half-life utility*, and *normalized distance-based performance measure* are discussed in more detail by Herlocker et al. (2004).

Challenges and Limitations

This section, presents some of the common hurdles in deploying recommender systems, as well as some research directions that address them.

Sparsity: Stated simply, most users do not rate most items and, hence, the user ratings matrix is typically very sparse. This is a problem for collaborative filtering systems, since it decreases the probability of finding a set of users with similar ratings. This problem often occurs when a system has a very high item-to-user ratio, or the system is in the initial stages of use. This issue can be mitigated by using additional domain information (Melville et al. 2002; Su et al. 2007) or making assumptions about the data generation process that allows for high-quality imputation (Su et al. 2008).

The Cold-Start Problem: New items and new users pose a significant challenge to recommender systems. Collectively these problems are referred to as the *cold-start problem* (Schein et al. 2002). The first of these problems arises in collaborative filtering systems, where an item cannot be recommended unless some user has rated it before. This issue applies not only to new items, but also to obscure items, which is particularly detrimental to users with eclectic tastes. As such the *new-item problem* is also often referred to as the *first-rater problem.* Since content-based approaches (Mooney and Roy 2000; Pazzani and Billsus 1997) do not rely on ratings from other users, they can be used to produce recommendations for *all* items, provided attributes of the items are available. In fact, the content-based predictions of similar users can also be used to further improve predictions for the active user (Melville et al. 2002).

The *new-user problem* is difficult to tackle, since without previous preferences of a user it is not possible to find similar users or to build a content-based profile. As such, research in this area has primarily focused on effectively selecting items to be rated by a user so as to rapidly improve recommendation performance with the least user feedback. In this setting, classical techniques from ▶ active learning can be leveraged to address the task of item selection (Harpale and Yang 2008; Jin and Si 2004).

Fraud: As recommender systems are being increasingly adopted by commercial websites, they have started to play a significant role in affecting the profitability of sellers. This has led to many unscrupulous vendors engaging in different forms of fraud to game recommender systems for their benefit. Typically, they attempt to inflate the perceived desirability of their own products (*push attacks*) or lower the ratings of their competitors (*nuke attacks*). These types of attack have been broadly studied as *shilling attacks* (Lam and Riedl 2004) or *profile injection attacks* (Burke et al. 2005). Such attacks usually involve setting up dummy profiles, and assume different amounts of knowledge about the system. For instance, the *average attack* (Lam and Riedl 2004) assumes knowledge of the average rating for each item; and the attacker assigns values randomly distributed around this average, along with a high rating for the item being *pushed.* Studies have shown that such attacks can be quite detrimental to predicted ratings, though *item-based* collaborative filtering tends to be more robust to these attacks (Lam and Riedl 2004). Obviously, content-based methods, which only rely on a users past ratings, are unaffected by profile injection attacks.

While pure content-based methods avoid some of the pitfalls discussed above, collaborative filtering still has some key advantages over them. Firstly, CF can perform in domains where there is not much content associated with items, or where the content is difficult for a computer to analyze, such as ideas, opinions, etc. Secondly, a CF system has the ability to provide serendipitous recommendations, that is, it can recommend items that are relevant to the user, but do not contain content from the user's profile.

Recommended Reading

Good surveys of the literature in the field can be found in Adomavicius and Tuzhilin (2005), Su (2009), Bell et al. (2009). For extensive empirical comparisons on variations of Collaborative Filtering refer to Breese et al. (1998), Herlocker et al. (1999), and Sarwar et al. (2001).

Adomavicius G, Tuzhilin A (2005) Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. IEEE Trans Knowl Data Eng 17(6):734–749

Balabanovic M, Shoham Y (1997) Fab: content-based, collaborative recommendation. Commun Assoc Comput Mach 40(3):66–72

Basu C, Hirsh H, Cohen W (July 1998) Recommendation as classification: using social and content-based information in recommendation. In: Proceedings of the fifteenth national conference on artificial intelligence (AAAI-98), Madison, pp 714–720

Bell R, Koren Y, Volinsky C (2009) Matrix factorization techniques for recommender systems. IEEE Comput 42(8):30–37

Billsus D, Pazzani MJ (1998) Learning collaborative information filters. In: Proceedings of the fifteenth international conference on machine learning (ICML-98), Madison. Morgan Kaufmann, San Francisco, pp 46–54

Breese JS, Heckerman D, Kadie C (1998) Empirical analysis of predictive algorithms for collaborative filtering. In: Proceedings of the fourteenth conference on uncertainty in artificial intelligence, Madison

Burke R, Mobasher B, Bhaumik R, Williams C (2005) Segment-based injection attacks against collaborative filtering recommender systems. In: ICDM '05: proceedings of the fifth IEEE international conference on data mining, Houston. IEEE Computer Society, Washington, DC, pp 577–580

Candès EJ, Tao T (2009) The power of convex relaxation: near-optimal matrix completion. IEEE Trans Inf Theory 56(5):2053–2080

Claypool M, Gokhale A, Miranda T (1999) Combining content-based and collaborative filters in an online newspaper. In: Proceedings of the SIGIR-99 workshop on recommender systems: algorithms and evaluation, Berkeley

Cotter P, Smyth B (2000) PTV: intelligent personalized TV guides. In: Twelfth conference on innovative applications of artificial intelligence, Austin, pp 957–964

Goldberg D, Nichols D, Oki B, Terry D (1992). Using collaborative filtering to weave an information tapestry. Commun Assoc Comput Mach 35(12): 61–70

Good N, Schafer JB, Konstan JA, Borchers A, Sarwar B, Herlocker J et al (1999) Combining collaborative filtering with personal agents for better recommendations. In: Proceedings of the sixteenth national conference on artificial intelligence (AAAI-99), Orlando, pp 439–446

Harpale AS, Yang Y (2008) Personalized active learning for collaborative filtering. In: SIGIR '08: proceedings of the 31st annual international ACM SIGIR conference on research and development in information retrieval, Singapore. ACM, New York, pp 91–98

Herlocker J, Konstan J, Borchers A, Riedl J (1999) An algorithmic framework for performing collaborative filtering. In: Proceedings of 22nd international ACM SIGIR conference on research and development in information retrieval, Berkeley. ACM, New York, pp 230–237

Herlocker JL, Konstan JA, Terveen LG, Riedl JT (2004) Evaluating collaborative filtering recommender systems. ACM Trans Inf Syst 22(1): 5–53

Hofmann T (1999) Probabilistic latent semantic analysis. In: Proceedings of the fifteenth conference on uncertainty in artificial intelligence, Stockholm, 30 July–1 Aug 1999. Morgan Kaufmann

Hofmann T (2004) Latent semantic analysis for collaborative filtering. ACM Trans Inf Syst 22(1): 89–115

Jin R, Si L (2004) A Bayesian approach toward active learning for collaborative filtering. In: UAI '04: proceedings of the 20th conference on uncertainty in artificial intelligence, Banff. AUAI Press, Arlington, pp 278–285

Lam SK, Riedl J (2004) Shilling recommender systems for fun and profit. In: WWW '04: proceedings of the 13th international conference on World Wide Web, New York. ACM, New York, pp 393–402

Lang K (1995) NewsWeeder: learning to filter netnews. In: Proceedings of the twelfth international conference on machine learning (ICML-95), Tahoe City. Morgan Kaufmann, San Francisco, pp 331–339. ISBN 1-55860-377-8.

Lee DD, Seung HS (1999) Learning the parts of objects by non-negative matrix factorization. Nature 401:788

Linden G, Smith B, York J (2003) Amazon.com recommendations: item-to-item collaborative filtering. IEEE Internet Comput 7(1):76–80

Melville P, Mooney RJ, Nagarajan R (2002) Content-boosted collaborative filtering for improved recommendations. In: Proceedings of the eighteenth national conference on artificial intelligence (AAAI-02), Edmonton, pp 187–192

Mooney RJ, Roy L (2000) Content-based book recommending using learning for text categorization. In: Proceedings of the fifth ACM conference on digital libraries, San Antonio, pp 195–204

Nakamura A, Abe N (1998) Collaborative filtering using weighted majority prediction algorithms. In: ICML '98: proceedings of the fifteenth international conference on machine learning, Madison. Morgan Kaufmann, San Francisco, pp 395–403

Pan R, Scholz M (2009) Mind the gaps: weighting the unknown in large-scale one-class collaborative filtering. In: 15th ACM SIGKDD conference on knowledge discovery and data mining (KDD), Paris

Pazzani MJ (1999) A framework for collaborative, content-based and demographic filtering. Artif Intell Rev 13(5–6):393–408

Pazzani MJ, Billsus D (1997) Learning and revising user profiles: the identification of interesting web sites. Mach Learn 27(3):313–331

Popescul A, Ungar L, Pennock DM, Lawrence S (2001) Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In: Proceedings of the seventeenth conference on uncertainity in artificial intelligence. University of Washington, Seattle

Recht B (2009, to appear) A simpler approach to matrix completion. J Mach Learn Res

Rennie J, Srebro N (2005) Fast maximum margin matrix factorization for collaborative prediction. In: International conference on machine learning, Bonn

Resnick P, Iacovou N, Sushak M, Bergstrom P, Reidl J (1994a) GroupLens: an open architecture for collaborative filtering of netnews. In: Proceedings of the 1994 computer supported cooperative work conference, New York. ACM, New York

Resnick P, Neophytos I, Bergstrom P, Mitesh S, Riedl J (1994b) Grouplens: an open architecture for collaborative filtering of netnews. In: CSCW94 – conference on computer supported cooperative work, Chapel Hill. Addison-Wesley, pp 175–186

Sarwar B, Karypis G, Konstan J, Reidl J (2001) Item-based collaborative filtering recommendation algorithms. In: WWW '01: proceedings of the tenth international conference on World Wide Web, Hong Kong. ACM, New York, pp 285–295

Schein AI, Popescul A, Ungar LH, Pennock DM (2002) Methods and metrics for cold-start recommendations. In: SIGIR '02: proceedings of the 25th annual international ACM SIGIR conference on research and development in information retrieval, Tampere. ACM, New York, pp 253–260

Soboroff I, Nicholas C (1999) Combining content and collaboration in text filtering. In: Joachims T (ed) Proceedings of the IJCAI'99 workshop on machine learning in information filtering, Stockholm, pp 86–91

Srebro N, Jaakkola T (2003) Weighted low-rank approximations. In: International conference on machine learning (ICML), Washington, DC

Su X, Khoshgoftaar TM (2009) A survey of collaborative filtering techniques. Adv Artif Intell 2009: 1–20

Su X, Greiner R, Khoshgoftaar TM, Zhu X (2007) Hybrid collaborative filtering algorithms using a mixture of experts. In: Web intelligence, pp 645–649

Su X, Khoshgoftaar TM, Zhu X, Greiner R (2008) Imputation-boosted collaborative filtering using machine learning classifiers. In: SAC '08: proceedings of the 2008 ACM symposium on applied computing. ACM, New York, pp 949–950

Record Linkage

Peter Christen[1] and William E. Winkler[2]
[1]Research School of Computer Science, The Australian National University, Canberra, ACT, Australia
[2]US Census Bureau, Suitland, MD, USA

Abstract

Many data mining and machine learning projects require information from various data sources to be integrated and linked before they can be used for further analysis. A crucial task of such data integration is to identify which records refer to the same real-world entities across databases when no common entity identifiers are available and when records can contain errors and variations. This process of record linkage therefore has to rely upon the attributes that are available in the databases to be linked. For databases that contain personal information, for example, of customers, taxpayers, or patients, these are commonly their names, addresses, phone numbers, and dates of birth.

To improve the scalability of the linkage process, blocking or indexing techniques are commonly applied to limit the comparison of records to pairs or groups that likely correspond to the same entity. Records are compared using a variety of comparison functions, most commonly approximate string comparators that account for typographical errors and variations in textual attributes.

The compared records are then classified into matches, non-matches, and potential matches, depending upon the decision model used. If training data in the form of true matches and non-matches are available, supervised classification techniques can be employed. However, in many practical record linkage applications, no ground truth data are available, and therefore unsupervised approaches are required. An approach known as probabilistic record linkage is commonly employed. In this article we provide an overview of record linkage with an emphasis on the classification aspects of this process.

Synonyms

Authority control; Citation or reference matching (when applied to bibliographic data); Co-reference resolution; Data linkage; Data matching; Data reconciliation; Deduplication or duplicate detection (when applied to one database only); Entity resolution; Field scrubbing; Identity uncertainty; List washing; Merge-purge; Object identification; Object matching; Reference reconciliation

Definition

Identifying and linking records that correspond to the same real-world entity in one or more databases is an increasingly important task in many data mining and machine learning projects. The aim of record linkage is to compare records within one (known as *deduplication*) or across two databases and classify the compared pairs of records as *matches* (pairs where both records are assumed to refer to the same real-world entity) and *non-matches* (pairs where the two records are assumed to refer to different entities).

Formally, let us consider two databases (or files), $\mathbf{A}$ and $\mathbf{B}$, and record pairs in the product space $\mathbf{A} \times \mathbf{B}$ (for the deduplication of a single database $\mathbf{A}$, the product space is $\mathbf{A} \times \mathbf{A}$). The aim of record linkage is to classify these record pairs into the classes of *matches* (links) and *non-matches* (non-links) (Christen 2012). Depending upon the decision model used (Fellegi and Sunter 1969; Herzog et al. 2007), a third class of *potential matches* (potential links) might be used. These are difficult to classify record pairs that will need to be manually assessed and classified as matches or non-matches in a manual clerical review process.

Each record pair in $\mathbf{A} \times \mathbf{B}$ is assumed to correspond to either a *true match* or a *true non-match*. The space $\mathbf{A} \times \mathbf{B}$ is therefore partitioned into the set M of true matches and the set U of true non-matches. The objective of record linkage is to correctly classify record pairs from M into the class of matches and pairs from U into the class of non-matches.

Motivation and Background

Increasingly, information systems and data mining projects require data from multiple sources to be integrated and linked in order to improve data quality, enrich existing data sources, or facilitate data analysis that is not feasible on an individual database. Compared to analyzing databases in isolation, the analysis of data linked across disparate sources, either within a single or between different organizations, can lead to much improved benefits. Integrated data can also allow types of analyses that are not feasible on individual databases, for example, the detection of fraud or terrorism suspects through the analysis of certain suspicious patterns of activities or the identification of adverse drug reactions in particular patient groups (Christen 2012). Record linkage has been employed in a wide range of domains as we discuss in section "Applications" below.

In most cases the databases to be linked (or deduplicated) do not contain unique entity identifiers or keys. Therefore, attributes (fields) that are common across the databases need to be used to identify similar records that likely correspond to the same entity. If the databases contain information about people, then these common attributes can be names, addresses, dates of birth, and other partially identifying personal details.

However, often the quality of such information is low, as personal details can be entered or recorded wrongly, be incomplete, or be out of date. Record linkage based on "dirty data" is challenging as ambiguities, errors, variations, and value changes can lead to both false matches (record pairs wrongly classified as referring to the same entity) and false non-matches (missed true matching pairs of records classified as non-matches).

The term record linkage was used in 1946 by Halbert Dunn to describe the idea of assembling a *book of life* for all individuals in the world (Dunn 1946). Each such book would begin with a birth record and end with a death record and in between would contain marriage and divorce records, as well as records about a person's contacts with the health and social security systems. Dunn realized that having such books of life for a full population would provide a wealth of information that would allow governments to improve national statistics, better plan services, and also help to identify individuals.

The first computer-based linkage techniques were proposed in the 1950s and early 1960s by Howard Newcombe et al. (1959), who also developed the basic ideas of the successful *probabilistic record linkage* approach described in the following section. Based on Newcombe's ideas, Fellegi and Sunter in 1969 published their seminal paper on a theory for probabilistic record linkage (Fellegi and Sunter 1969). They proved that an optimal probabilistic decision rule can be found under the assumption of independence of the attributes used in the comparison of records. This influential work has been the basis for many record linkage systems and software products, and it is still widely used today.

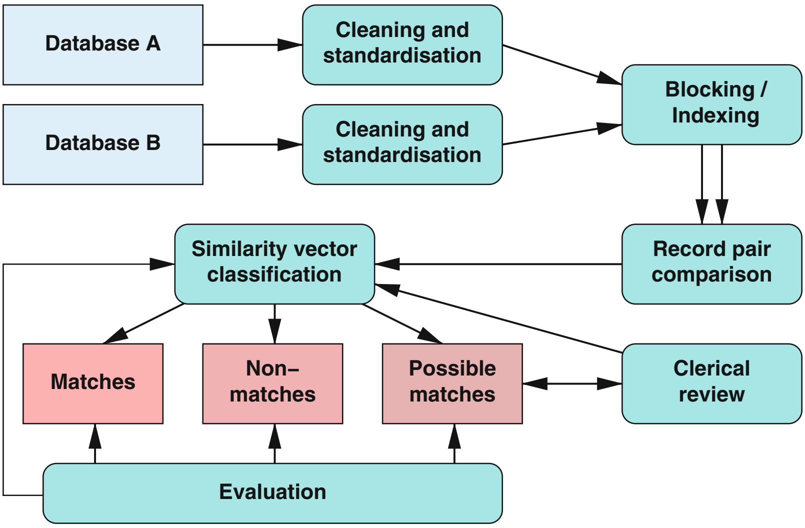
Theory/Solution

We now describe the steps involved in the record linkage process, followed by a more detailed discussion of techniques used to classify record pairs into matches and non-matches. Note that for the deduplication of a single database, all steps of the record linkage process are still applicable.

The Record Linkage Process

Figure 1 provides an overview of the steps involved in the general record linkage process. We assume two databases that contain details about the same types of entities (such as people, businesses, scientific publications, and so on). The first step of *data cleaning and standardization* is important to convert the input data into the same format so they are more suitable for comparison. This step involves, for example, converting all letters into lower or upper case, removing

Record Linkage, Fig. 1 The general process of linking two databases (As adapted from Christen 2012)



certain punctuations and words, and splitting attributes into specific fields (such as title, first name, middle name, and last name for personal names).

The second step of *blocking* or *indexing* is aimed at reducing the number of record pairs that need to be compared from the full pairwise comparison space of $|\mathbf{A}| \times |\mathbf{B}|$, where $|\cdot|$ is the number of records in a database. The idea is to only compare records in detail that likely refer to matches. This is accomplished by splitting the databases into blocks according to some criteria and then only comparing records in the same block across the two databases. An example criteria can, for example, be a post- or zip code attribute, resulting in only records being compared that have the same post- or zip code value. Various such blocking or indexing techniques have been developed in the past few decades (Christen 2012).

In the *comparison* step, candidate record pairs generated in the blocking/indexing step are then compared in detail using a variety of attribute and record comparison functions. As many attributes used in record linkage to compare records contain textual values (like names and addresses), approximate string comparison functions such as Jaro-Winkler or edit distance are commonly used (Christen 2012). Specific comparison functions have also been developed for values such as ages, dates, or phone numbers (Christen 2012). Generally all these comparison functions return a numerical similarity, s, that is normalized in $0 \leq s \leq 1$, with $s = 1$ if two attribute values are the same (like "geoff" and "geoff"), $s = 0$ if they are completely different (like "claude" and "geoff"), and $0 < s < 1$ if they are somehow similar (like "geoff" and "jeff"). For each compared record pair, a similarity vector (also known as weight vector) is formed that contains the similarities of all compared attributes of that pair.

In the *classification* step (as we discuss in more details below), each compared candidate record pair is classified into one of the classes of matches, non-matches, and possibly also potential matches, depending upon the decision model used (Fellegi and Sunter 1969; Herzog et al. 2007). Various classification techniques (both supervised and unsupervised) have been developed in the past nearly five decades.

If candidate record pairs have been classified into potential matches, a manual *clerical review* process is required to decide their final match status (match or non-match). These manual classifications can flow back into the classification model when an active learning approach is employed. Several active learning approaches have been developed for record linkage (Christen 2012).

In the final *evaluation* step, the complexity, completeness, and quality of the linked records are evaluated using a variety of measures (Christen 2012). The complexity of a linkage can be measured by the number of candidate record pairs generated by an indexing or blocking technique. Measuring completeness and linkage quality requires truth data in the form of known true matching and non-matching record pairs. Linkage quality is generally measured using precision and recall, while completeness is similar to recall but measures how many of all known true matches are included in the set of candidate record pairs (i.e., how many true matches are not removed in the indexing/blocking step).

Record Linkage Model of Fellegi and Sunter

Fellegi and Sunter (1969) provided a formal mathematical model for ideas that had been introduced by Newcombe et al. (1959). Fellegi and Sunter (1969) also provided ways of estimating key parameters without training data. Generally, training data have not been available for most record linkage applications.

Following the notation used in section "Definition" above, Fellegi and Sunter, making rigorous concepts introduced by Newcombe et al. (1959), considered ratios of probabilities of the form:

$$R = \frac{P(\gamma \in \Gamma|M)}{P(\gamma \in \Gamma|U)}, \tag{1}$$

where γ is an arbitrary agreement pattern in a comparison space Γ. For instance, Γ might consist of eight patterns representing simple agreement or not on the largest name component, street name, and street number (Herzog et al. 2007). Alternatively, each $\gamma \in \Gamma$ might additionally account for the relative frequency with which specific values of name components such as "Smith", "Zabrinsky", "AAA", and "Capitol" occur. The ratio R or any monotonically increasing function of it (such as the natural log) is referred to as a *matching weight* (or score). The decision rule is given by:

$$\begin{aligned} &\text{If } R \geq T_\mu, && \text{then designate pair as a match.} \\ &\text{If } T_\lambda < R < T_\mu, && \text{then designate pair as a possible match} \\ & && \text{and hold for clerical review.} \\ &\text{If } R \leq T_\lambda, && \text{then designate pair as a non-match.} \end{aligned} \tag{2}$$

The cutoff thresholds T_λ and T_μ are determined by a priori error bounds on false matches and false non-matches. The thresholds are often called lower and upper cutoffs. Rule (2) agrees with intuition. If $\gamma \in \Gamma$ consists primarily of agreements, then it is intuitive that $\gamma \in \Gamma$ would be more likely to occur among matches than non-matches, and the ratio (1) would be large. On the other hand, if $\gamma \in \Gamma$ consists primarily of disagreements, then the ratio (1) would be small. Rule (2) partitions the set $\gamma \in \Gamma$ into three disjoint subregions. The region $\{T_\lambda < R < T_\mu\}$ is referred to as the no-decision region or clerical review region (Herzog et al. 2007). In some situations, resources are available to review pairs clerically.

Figure 2 provides an illustration of the curves of log frequency versus log weight for matches and non-matches, respectively. The data used in Fig. 2 are based on information obtained while matching name and address files from one of the sites for the 1988 US Dress Rehearsal Census. The clerical review region consists of individuals within the same household that are missing both name and age. Figure 2 shows hypothetical cutoff thresholds that we denote with the symbols L (lower) and U (upper) in this figure, respectively.

Learning Parameters via the Methods of Fellegi and Sunter

Fellegi and Sunter (1969) were the first to provide very general methods for computing the probabilities in the ratio (1). As the methods are useful, we describe what they introduced and then show how the ideas led into more general methods that can be used for unsupervised learning (i.e., without training data) in a large number of situations.

Fellegi and Sunter observed several things. Firstly,

$$P(S) = P(S|M)P(M) + P(S|U)P(U) \tag{3}$$

for any set S of pairs in $\mathbf{A} \times \mathbf{B}$. The probability on the left can be computed directly from the set of pairs. In Equation (3), M and U are restricted to S. Secondly, if sets A^x represent simple agreement/disagreement, under the conditional independence assumption (**CI**) (i.e., naive Bayes), we obtain

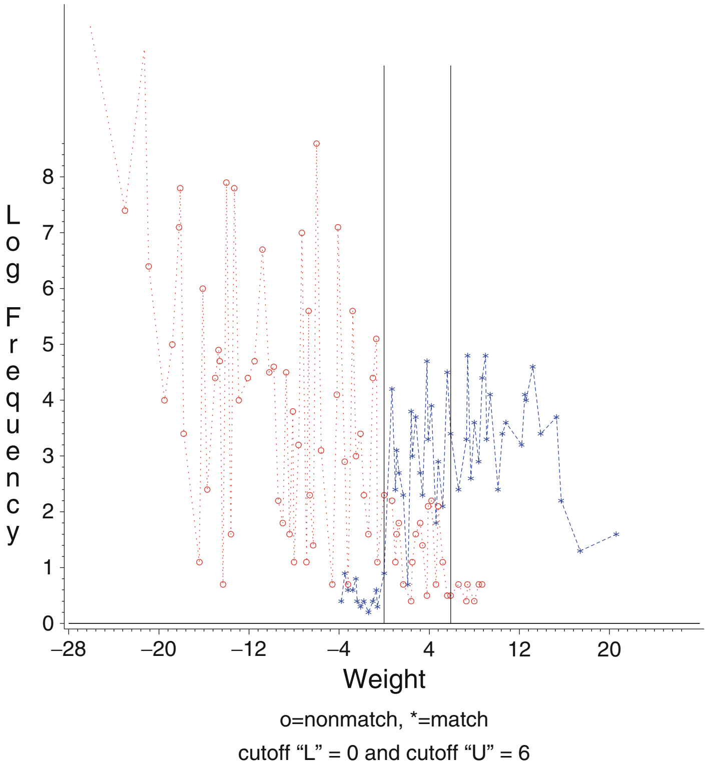
$$\begin{aligned} &P(A_1^x \cap A_2^x \cap A_3^x|D) \\ &\quad = P(A_1^x|D)P(A_2^x|D)P(A_3^x|D). \end{aligned} \tag{4}$$

Here D is either M or U. Then (3) and (4) provide seven equations and seven unknowns (as x represents agree or disagree) that yield quadratic equations that they solved. Equation (or set of equations) (4) can be expanded to K fields.

The expectation-maximization (EM) algorithm (Dempster et al. 1977) can be used to estimate the probabilities in Eqs. (3) and (4) when there are more than K fields or when condition (**CI**) may not hold.

For the 1990 US Decennial Census, Winkler (1988) introduced an EM algorithm that found the best naive Bayes approximation of a general Bayes net model where interactions between fields were accounted for. This type of EM was necessary because "optimal" parameters were used for each of the ~500 regions into which the

Record Linkage, Fig. 2 Log frequency versus weight, matches and non-matches combined



USA was divided, the entire matching operations needed to be completed in less than 6 weeks to provide estimates required under the US law, and it was impossible to obtain training data.

Herzog et al. (2010) provide many of the details of the EM procedures used in the 1990 US Decennial Census production matching systems that we do not cover here. We provide two highlights that were in Herzog et al. (2010). Firstly, the EM algorithm in this particular application was able to adapt automatically to increasing missing data. During 1 week in one of seven processing offices, it was discovered that the clerical review region increased significantly. Upon follow-up, it was determined that two clerks had managed to bypass keypunch edits on the year-of-birth field, and all records keyed by them disagreed on the computed age. Age and first name were the only fields that would allow distinguishing true matches within households.

Secondly, the probabilities from the unsupervised learning yielded better matching results than results from an iterative refinement procedure (a type of *active learning*) that was in widespread use for matching. In the iterative refinement procedure, a subset of clerical pairs were followed up to determine matches and nonmatches, the matching probabilities were reestimated, an additional set of clerical pairs were followed up, and parameters were reestimated, with the entire interactive learning procedure being repeated on the order of five cycles until the matching probabilities stabilized.

Superficially, the EM algorithm (Winkler 1988) considers different orderings of the form

$$P(A^x_{\rho,1} \cap \cdots \cap A^x_{\rho,k} | D) \\ = \Pi^K_{i=1} P(A^x_{\rho,i} | A^x_{\rho,i-1}, \cdots, A^x_{\rho,1}, D), \quad (5)$$

where ρ, i represents the ith entry in a permutation ρ of the integers 1 thru K. The greater generality of (5) in comparison to (4) can yield better fits to the data. It can be reasonably assumed that the EM algorithm under the conditional independence assumption (as the actual computational methods work) simultaneously chooses the best permutation and the best parameters.

Because training data are seldom available and can be exceptionally expensive to obtain, some authors (Larsen and Rubin 2001) recommend *semi-supervised learning* where a small amount of judiciously chosen training data are combined with a large amount of unlabeled data for which true matching status is unknown. The semi-supervised methods generally outperform the unsupervised methods.

Some commercial record linkage software uses *rule-based* methods, which employ strategies such as if these three fields are the same in a pair of records, call the pair a designated match, designated link, or possible same entity. Ferrante and Boyd (2012), in a large comparison, showed that one rule-based commercial package was outperformed by one commercial package and several shareware packages that each applied variants of the Fellegi-Sunter model.

Applications

Record linkage has been used in a wide range of domains (Christen 2012; Herzog et al. 2007). In the following we briefly describe some example applications:

- **Linking personal data**: Traditionally the most common use of record linkage is to identify and link records about the same person across two databases. Examples of such linkages occur in national censuses (linking people between two census collections), in the health domain (linking patient records between different hospital and healthcare providers or over time with the aim to compile patient-oriented longitudinal data sets for public health studies), or between government agencies to, for example, identify people who commit welfare fraud.

  In the health domain, population informatics (Kum et al. 2014), the study of populations by linking and analyzing large databases that contain detailed information about a large proportion of individuals in a population (such as their health, education, financial, census, location, shopping, employment, or social networking records), has recently attracted increasing interest.
- **Deduplication of customer databases**: A common data quality problem for many businesses is that a customer might be recorded in their databases more than once due to address or name changes and variations. Such duplicates can incur significant costs for a business, for example, when sending out advertisement mail. The task of deduplicating a single database is in principle the same as when linking two databases. Each record in the database potentially needs to be compared with all others (indexing or blocking is generally also applied to speed up the deduplication process).
- **Linking historical population data**: The quantitative social sciences are currently seeing a shift toward the use of large-scale data collections for studying a diverse range of aspects of the human society. Often these are historical data such as census, birth, death, and marriage registries that span several decades (or even centuries) and that need to be linked to reconstruct historical populations (Bloothooft et al. 2015). The major challenges when linking such data include data quality (as such data have to be transcribed from hand-written forms, a process that is error-prone and labor-intensive), the dynamics of people's characteristics over time, and the complexity of roles and relationships for each individual as they change over time.
- **Consumer product comparison shopping**: With the increasing popularity of online comparison shopping Web sites, the challenging task of identifying which descriptions of products across diverse shopping sites correspond to the same real-world product has attracted interest from various domains. Compared to personal data, such as names and addresses, different consumer products might only be distinguishable by a single digit (such as the *Canon 600D* and *Canon 650D* digital cameras). To improve linkage quality in this domain, novel similarity calculation functions and machine learning approaches

that learn the characteristic features that distinguish consumer products have been developed (Christen 2012).

- **Linking bibliographic data**: Research is increasingly being published through online databases such as *Springer Link* or the *ACM Digital Library*. These databases facilitate a much faster dissemination of knowledge, and they allow government funding agencies to calculate numerical metrics to assess the impact of researchers, research groups, and even institutions. This requires to link all records of an individual researcher with high accuracy. A major challenge with bibliographic data is that there can be several researchers with the same name details in a database, some even working in the same research domain. Even if full given names are provided, it can be unclear if two publications were written by the same individual or not. Journal and conference names are also often abbreviated and do not follow standardized formats.

Future Directions

Most research in record linkage in the past decade has concentrated on improving either the scalability of the linkage process through the development of advanced indexing or blocking techniques (Christen 2012) or linkage quality by employing sophisticated classification techniques. Most of these techniques assume the databases to be linked are static and the linkage can be done off-line and in batch mode. In the following we summarize areas of ongoing research that aim to address various practical problems in record linkage.

- **Collective classification**: Traditional record linkage techniques classify each compared record pair individually (Herzog et al. 2007). This can lead to violations of transitivity (if record A is classified as a match with record B, and record B as a match with record C, then records A and C must also be a match). With traditional approaches, transitivity is often addressed in a post-linkage process (Christen 2012).

  Recently developed graph-based and collective *entity resolution* techniques (Bhattacharya and Getoor 2007) instead aim to find an overall optimal assignment of records to entities. These techniques take both attribute similarities and relationship information into account. They generally build a graph where nodes are records and edges connect records that have a similarity above a certain minimum threshold. The task then becomes one of splitting such a graph into individual subgraphs such that each subgraph contains the records of one entity only, and each entity is represented by one subgraph.

  While such techniques have been shown to achieve high linkage quality (mainly on bibliographic data), their computational complexity (quadratic or larger in the number of records to be linked) makes the application of these techniques to large-scale record linkage problems challenging. Furthermore, how to employ such collective linkage techniques in domains where only limited relational information is available (such as for data about people) is an open question.

- **Group linkage**: Related to the previous topic is the challenge of linking groups of records instead of individual pairs. Groups can, for example, represent the people in a household or family or the coauthors of a scientific publication. Group linkage (On et al. 2007) is generally a two-step process, where in the first step, individual record pairs are linked, followed by the linkage of groups using some form of bipartite graph matching. The challenges in group linkage occur when groups do not have the same number of members, when group membership changes over time, and when groups can split or merge, such as does happen in families and households.
- **Linking temporal data**: Most personal details of people change over time, such as their addresses, names, employments, and relationships. If records with such details have time stamps attached (such as the dates when the information was recorded), then considering such temporal information might help improve linkage quality (Li et al. 2011). For ex-

ample, if it is known that a certain proportion of people in a population move their address between two census collections 5 years apart, then less weight should be assigned to address similarities when the overall similarities between records are calculated. Some initial work has investigated how such adjustment of similarity weights can help improve overall linkage quality for bibliographic data (assuming authors change their institutions); however more research is needed to investigate if and how such techniques can be employed when, for example, linking census or health data.

- **Statistical analysis across multiple files**: Economists and demographers want to analyze $(\mathbf{X}, \mathbf{Y})$ – data where multivariate $\mathbf{X}$ is taken from one file **A** and multivariate $\mathbf{Y}$ is taken from another file **B**. The common data for linking the files are typically nonunique identifiers such as name, address, and date of birth. Lahiri and Larsen (2005) provide such a model that can adjust certain statistical analyses such as regression under modest assumptions for linkage error. Others have considered $(\mathbf{X}, \mathbf{Y})$ – data where $\mathbf{X}$ and $\mathbf{Y}$ are each composed of discrete or continuous data but under very strong assumptions.
- **Real-time linkage**: As services in the public and private sectors move online, organizations increasingly require real-time linkage in applications such as online identity verification based on personal details or web and document search where duplicates in the set of retrieved records or documents need to be identified. Compared to the batch linkage of two databases, real-time linkage considers a stream of query records that need to be linked in sub-second time with a potentially large database that contains entity records. Often these databases are also dynamic, where new records are added and existing records are modified.

  Novel indexing techniques are required that allow the efficient and effective retrieval of candidate records that are likely to be matching with a query record (Ramadan et al. 2015), as well as fast classification and ranking techniques that are adaptive to changes in the underlying entity database.
- **Privacy-preserving record linkage**: In many application domains, record linkage relies on personal details, such as names and addresses, to conduct the linkage. Privacy and confidentiality issues can be of great concern, especially when databases are linked between organizations. Many countries have privacy legislation that limits the sharing and use of personal information. Linking records, for example, between a private hospital and a government health department might therefore be limited or even prohibited.

  The past decade has seen the emergence of research that aims to develop techniques that allow the linking of databases across organizations while ensuring that no sensitive private or confidential information is being revealed (Vatsalan et al. 2013). Using encoding techniques such as one-way hashing and Bloom filters, and cryptographic approaches such as secure multiparty computation, privacy-preserving record linkage techniques encode records at the data sources in such ways that similarity calculations and approximate matching of string values are feasible, while still allowing the linkage of large databases in efficient and effective ways.

Cross-References

▶ Classification
▶ Data Preparation
▶ Entity Resolution
▶ Expectation Maximization Clustering
▶ Link Mining and Link Discovery
▶ Link Prediction
▶ Similarity Measures
▶ Unsupervised Learning

Recommended Reading

Bhattacharya I, Getoor L (2007) Collective entity resolution in relational data. ACM Trans Knowl Discov Data 1(1), 5-es, pp 1–35

Bloothooft G, Christen P, Mandemakers K, Schraagen M (2015) Population reconstruction. Springer, Cham

Christen P (2012) Data matching – concepts and techniques for record linkage, entity resolution, and duplicate detection. Data-centric systems and applications. Springer, Berlin/New York

Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J R Stat Soc Ser B 19:380–393

Dunn H (1946) Record linkage. Am J Publ Health 36(12):1412

Fellegi IP, Sunter AB (1969) A theory for record linkage. J Am Stat Assoc 64(328):1183–1210

Ferrante A, Boyd J (2012) A transparent and transportable methodology for evaluating data linkage software. J Biomed Inf 45(1):165–172

Herzog TN, Scheuren FJ, Winkler WE (2007) Data quality and record linkage techniques. Springer, New York/London

Herzog TN, Scheuren FJ, Winkler WE (2010) Record linkage. Wiley Interdiscip Rev Comput Stat 2(5): 535–543

Kum HC, Krishnamurthy A, Machanavajjhala A, Ahalt SC (2014) Social genome: putting big data to work for population informatics. IEEE Comput 47(1):56–63

Lahiri P, Larsen M (2005) Regression analysis with linked data. J Am Stat Assoc 100:222–230

Larsen MD, Rubin DB (2001) Iterative automated record linkage using mixture models. J Am Stat Assoc 96(453):32–41

Li P, Dong XL, Maurino A, Srivastava D (2011) Linking temporal records. The VLDB conference was in Seattle, WA. In: Proceedings of the VLDB endowment, Seattle, vol 4, issue 11

Newcombe H, Kennedy J, Axford S, James A (1959) Automatic linkage of vital records. Science 130(3381):954–959

On BW, Koudas N, Lee D, Srivastava D (2007) Group linkage. In: IEEE international conference on data engineering, Istanbul, pp 496–505

Ramadan B, Christen P, Liang H, Gayler RW (2015) Dynamic sorted neighborhood indexing for real time entity resolution. ACM J Data Inf Qual 6(4):15

Vatsalan D, Christen P, Verykios VS (2013) A taxonomy of privacy-preserving record linkage techniques. Elsevier Inf Syst 38(6):946–969

Winkler WE (1988) Using the EM algorithm for weight computation in the Fellegi-Sunter model of record linkage. The American Statistical Association that is located in Alexandria, VA publishes the proceedings. In: Proceedings of the section on survey research methods, New Orleans, Washington, pp 667–671

Recurrent Associative Memory

▶ Hopfield Network

Recursive Partitioning

▶ Divide-and-Conquer Learning

Reference Reconciliation

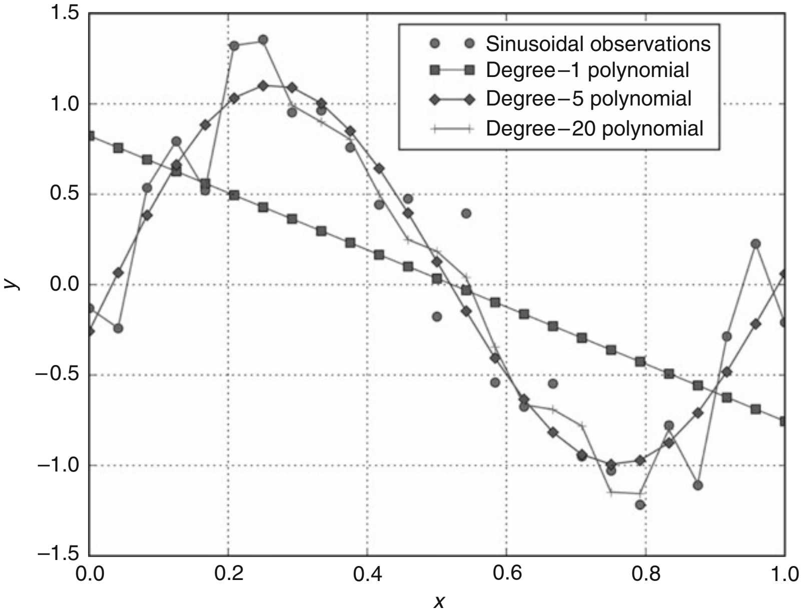
▶ Entity Resolution
▶ Record Linkage

Regression

Novi Quadrianto[1] and Wray L. Buntine[2,3]
[1]Department of Informatics, SMiLe CLiNiC, University of Sussex, Brighton, UK
[2]Statistical Machine Learning Program, NICTA, Canberra, ACT, Australia
[3]Faculty of Information Technology, Monash University, Clayton, VIC, Australia

Definition

Regression is a fundamental problem in statistics and machine learning. In regression studies, we are typically interested in inferring a real-valued function (called a regression function) whose values correspond to the mean of a dependent (or response or output) variable conditioned on one or more independent (or input) variables. Many different techniques for estimating this regression function have been developed, including parametric, semi-parametric, and nonparametric methods.



Regression, Fig. 1 Twentyfive data points (one-dimensional input x and output y variables) with a Gaussian-corrupted sinusoidal input–output relationship, $y = \sin(2\pi x) + \epsilon$ where ϵ is the normally distributed noise. The task is to learn the functional relationship between x and y. Various lines show the inferred relationship based on a linear regression model with polynomial basis functions having various degrees

Motivation and Background

Assume that we are given a set of data points sampled from an underlying but unknown distribution, each of which includes input x and output y. An example is given in Fig. 1. The task of regression is to learn a hidden functional relationship between x and y from observed and possibly noisy data points. In Fig. 1, the input–output relationship is a Gaussian-corrupted sinusoidal relationship, that is, $y = \sin(2\pi x) + \epsilon$ where ϵ is the normally distributed noise. Various lines show the inferred relationship based on a linear parametric regression model with polynomial basis functions. The higher the degree of the polynomial, the more complex is the inferred relationship, as shown in Fig. 1, as the function tries to better fit the observed data points.

While the most complex polynomial here is an almost perfect reconstruction of observed data points (it has "low bias"), it gives a very poor representation of the true underlying function $\sin(2\pi x)$ that can change significantly with the change of a few data points (it has "high variance"). This phenomenon is called the ▶ *bias-variance dilemma*, and selecting a complex model with too high a variance is called ▶ *overfitting*. Complex parametric models (like polynomial regression) lead to low bias estimators with a high variance, while simple models lead to low variance estimators with high bias. To sidestep the problem of trying to estimate or select the model complexity represented, for instance, by the degree of the polynomial, so-called nonparametric methods allow a rich variety of functions from the outset (i.e., a function class not finitely parameterizable) and usually provide a hyperparameter that tunes the regularity, curvature, or complexity of the function.

Theory/Solution

Formally, in a regression problem, we are interested in recovering a functional dependency $y_i = f(x_i) + \epsilon_i$ from N-observed training data points $\{(x_i, y_i)\}_{i=1}^N$, where $y_i \in \mathbb{R}$ is the noisy observed output at input location $x_i \in \mathbb{R}^d$. For ▶ linear regression, we represent the regression function $f()$ by a parameter $w \in \mathbb{R}^H$ in the form $f(x_i) := \langle \phi(x_i), w \rangle$ for H fixed basis functions $\{\phi_h(x_i)\}_{h=1}^H$. With general basis functions such as polynomials, exponentials, sigmoids, or even more sophisticated Fourier or wavelets bases, we can obtain a regression function which is nonlinear with regard to the input variables although still linear with regard to the parameters.

In regression, many more methods are possible. Some variations on these standard linear models are piecewise linear models, trees, and splines (roughly, piecewise polynomial models joined up smoothly) (Hastie et al. 2003). These are called semi-parametric models, because they have a linear parametric component as well as a nonparametric component.

Fitting

In general, regression fits a model to data using an objective function or quality criterion in a form such as

$$E(f) = \sum_{i=1}^{N} \epsilon(y_i, f(x_i)) \,,$$

where smaller $E(f)$ implies better quality. This might be derived as an error/loss function or as a negative log likelihood or log probability. The squared error function is the most convenient (leading to a least squares calculation), but many possibilities exist. In general, methods are distinguished by three aspects: (1) the representation of the function $f()$, (2) the form of the term $\epsilon(y_i, f(x_i))$, and (3) the penalty term discussed next.

Regularized/Penalized Fitting

The issue of overfitting, as mentioned already in the section Motivation and Background, is usually addressed by introducing a regularization or penalty term to the objective function. The regularized objective function is now in the form of:

$$E_{\text{reg}} = E(f) + \lambda R(f). \tag{1}$$

Here, $E(f)$ measures the quality of the solution for $f()$ on the observed data points, $R(f)$ penalizes complexity of $f()$, and λ is called the regularization parameter which controls the relative importance between the two. Measures of function curvature, for instance, can be used for $R(f)$. In standard ▶ support vector machines, the term $E(f)$ measures the hinge loss, and penalty $R(f)$ is the sum of squares of the parameters, also used in ridge regression (Hastie et al. 2003).

Bias-Variance Dilemma

As we have seen in the previous section, the introduction of the regularization term can help avoid overfitting. However, this raises the question of determining an optimal value for the regularization parameter λ. The specific choice of λ controls the bias-variance tradeoff (Geman et al. 1992).

Recall that we try to infer a latent regression function $f(x)$ based on N-observed training data points $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$. The notation $f(x; \mathcal{D})$ explicitly shows the dependence of f on the data $\mathcal{D}$. The mean squared error (MSE) which measures the effectiveness of f as a predictor of y is

$$\begin{aligned}&\mathbf{E}[(y - f(x;\mathcal{D}))^2 | x, \mathcal{D}]\\ &= \mathbf{E}[(y - \mathbf{E}[y|x])^2 | x, \mathcal{D}] + (f(x;\mathcal{D}) - \mathbf{E}[y|x])^2\end{aligned} \tag{2}$$

where $\mathbf{E}[.]$ means expectation with respect to a conditional distribution $p(y|x)$. The first term of (2) does not depend on $f(x; \mathcal{D})$, and it represents the intrinsic noise on the data. The MSE of f as an estimator of the regression $\mathbf{E}[y|x]$ is

$$\mathbf{E}_{\mathcal{D}}[(f(x;\mathcal{D}) - \mathbf{E}[y|x])^2] \tag{3}$$

where $\mathbf{E}_{\mathcal{D}}$ means expectation with respect to the training set $\mathcal{D}$. The estimation error in (3) can be decomposed into a bias and a variance terms, that is,

$$
\begin{aligned}
\mathbf{E}_{\mathcal{D}}[(f(x;\mathcal{D})-\mathbf{E}[y|x])^2] &= \mathbf{E}_{\mathcal{D}}[(f(x;\mathcal{D})\\
&\quad -\mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})]+\mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})]-\mathbf{E}[y|x])^2]\\
&= \mathbf{E}_{\mathcal{D}}[(f(x;\mathcal{D})-\mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})])^2]\\
&\quad +(\mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})]-\mathbf{E}[y|x])^2+2\mathbf{E}_{\mathcal{D}}[(f(x;\mathcal{D})\\
&\quad -\mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})])](\mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})]-\mathbf{E}[y|x])\\
&= \mathbf{E}_{\mathcal{D}}[(f(x;\mathcal{D})-\mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})])^2]\\
&\quad +(\mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})]-\mathbf{E}[y|x])^2\\
&= \text{variance}+\text{bias}^2.
\end{aligned}
$$

The bias term measures the difference between the average predictor over all datasets and the desired regression function. The variance term measures the adaptability of the predictor to a particular dataset. There is a tradeoff between the bias and variance contributions to the estimation error, with very flexible models having low bias but high variance (overfitting) and relatively rigid models having low variance but high bias (underfitting). Typically, variance is reduced through "smoothing," that is, an introduction of the regularization term. This, however, will introduce bias as peaks and valleys of the regression function will be blurred. To achieve an optimal predictive capability, an estimator with the best balance between bias and variance is chosen by varying the regularization parameter λ. It is crucial to note that bias-variance decomposition albeit powerful is based on averages of datasets; however, in practice only a single dataset is observed. In this regard, a Bayesian treatment of regression, such as Gaussian process regression which will avoid overfitting problem of maximum likelihood and which will also lead to automatic methods of determining model complexity using the training data alone, could be an attractive alternative.

Nonparametric Regression

In the parametric approach, an assumption on the mathematical form of the functional relationship between input x and output y such as linear, polynomial, exponential, or combination of them needs to be chosen a priori. Subsequently, parameters are placed on each of the chosen forms and the optimal values learned from the observed data. This is restrictive both in the fixed functional form and in the ability to vary the model complexity. Nonparametric approaches try to derive the functional relationship directly from the data, that is, they do not parameterize the regression function.

▶ Gaussian Processes for regression, for instance, are well developed. Another approach is the *kernel method*, of which a rich variety exists (Hastie et al. 2003). These can be viewed as a regression variant of nearest neighbor classification where the function is made up of a local element for each data point:

$$f(x) = \frac{\sum_i y_i K_\lambda(x_i,x)}{\sum_i K_\lambda(x_i,x)},$$

where the function $K_\lambda(x_i,)$ is a nonnegative "bump" in x space centered at its first argument with diameter approximately given by λ. Thus, the function has a variable contribution from each data point and λ controls the bias-variance tradeoff.

Generalized Linear Models

The previous discussion about regression focuses on continuous output/dependent variables. While this type of regression problem is ubiquitous, there are however some interests in cases of *restricted* output variables:

1. The output variable consists of two categories (called *binomial* regression).
2. The output variable consists of more than two categories (called *multinomial* regression).
3. The output variable consists of more than two categories which can be ordered in a meaningful way (called *ordinal* regression). and
4. The output variable is a count of the repetition of the occurrence of an event (called *poisson* regression).

Nelder and Wedderburn (1972) introduced the generalized linear model (GLM) by allowing the linear model to be related to the output variables via a link function. This is a way to unify different cases of response variables under one

Regression, Table 1 A table of various link functions associated with the assumed distribution on the output variable

Distribution of dependent variable	Name	Link function
Gaussian	Identity link	$g(\mu) = \mu$
Poisson	Log link	$g(\mu) = \log(\mu)$
Binomial multinomial	Logit link	$g(\mu) = \log\left(\frac{\mu}{1-\mu}\right)$
Exponential gamma	Inverse link	$g(\mu) = \mu^{-1}$
Inverse Gaussian	Inverse squared link	$g(\mu) = \mu^{-2}$

framework, each only differs in the choice of the link function. Specifically, in GLM, each output variable is assumed to be generated from the exponential family of distributions. The mean of this distribution depends on the input variables through

$$\mathbf{E}[y] = g(\mu) = w_0 + w_1\phi_1(x_i) + \ldots + w_D\phi_D(x_i), \tag{4}$$

where $g(\mu)$ is the link function (Table 1). The parameters of the generalized linear model can then be estimated by the maximum likelihood method, which can be found by iterative reweighted least squares (IRLS), an instance of the expectation maximization (EM) algorithm.

Other Variants of Regression

So far, we have focused on the problem of predicting a single output variable y from an input variable x. Some studies look at predicting multiple output variables simultaneously. The simplest approach for the *multiple outputs* problem would be to model each output variable with a different set of basis functions. The more common approach uses the same set of basis functions to model all of the output variables. Not surprisingly, the solution to the multiple outputs problem decouples into independent regression problems with shared basis functions.

For some other studies, the focus of regression is on computing several regression functions corresponding to various percentage points or quantiles (instead of the mean) of the conditional distribution of the dependent variable given the independent variables. This type of regression is called *quantile* regression (Koenker 2005). The sum of tilted absolute loss (called pinball loss) is being optimized for this type of regression. Quantile regression has many important applications within econometrics, data mining, social sciences, and ecology, among other domains.

Instead of inferring one regression function corresponding to the mean of a response variable, k regression functions can be computed with the assumption that the response variable is generated by a mixture of k components. This is called the *mixture of regressions* problem (Gaffney and Smyth 1999). Applications include trajectory clustering, robot planning, and motion segmentation.

Another important variant is the *heteroscedastic* regression model where the noise variance on the data is a function of the input variable x. The Gaussian process framework can be used conveniently to model this noise-dependent case by introducing a second Gaussian process to model the dependency of noise variance on the input variable (Goldberg et al. 1998). There are also attempts to make the regression model more robust to the presence of a few problematic data points called outliers. The sum of absolute loss (instead of the sum of squared loss) or student's t-distribution (instead of Gaussian distribution) can be used for *robust* regression.

Cross-References

- ▶ Gaussian Processes
- ▶ Linear Regression
- ▶ Support Vector Machines

Recommended Reading

Machine learning textbooks such as Bishop (2006), among others, introduce different regression models. For a more statistical introduction including an extensive overview of the many different semi-parametric methods and non-parametric methods such as kernel methods, see Hastie et al. (2003). For a coverage of key statistical issues including nonlinear regression, identifiability, measures of curvature, autocorrelation, and such, see Seber and Wild (1989). For a large variety of built-in regression techniques, refer to R (http://www.r-project.org/).

Bishop C (2006) Pattern recognition and machine learning. Springer, New York
Gaffney S, Smyth P (1999) Trajectory clustering with mixtures of regression models. In: ACM SIGKDD, vol 62. ACM, New York, pp 63–72
Geman S, Bienenstock E, Doursat R (1992) Neural networks and the bias/variance dilemma. Neural Comput 4:1–58
Goldberg P, Williams C, Bishop C (1998) Regression with input-dependent noise: a Gaussian process treatment. In: Neural information processing systems, vol 10. MIT
Hastie T, Tibshirani R, Friedman J (Corrected ed) (2003) The elements of statistical learning: data mining, inference, and prediction. Springer, New York
Koenker R (2005) Quantile regression. Cambridge University Press, Cambridge
Nelder JA, Wedderburn RWM (1972) Generalized linear models. J R Stat Soc: Ser A 135: 370–384
Seber G, Wild C (1989) Nonlinear regression. Wiley, New York

Regression Trees

Luís Torgo
University of Porto, Porto, Portugal

Synonyms

Decision trees for regression; Piecewise constant models; Tree-based regression

Definition

Regression trees are supervised learning methods that address multiple regression problems. They provide a tree-based approximation $\hat{f}$, of an unknown regression function $Y = f(\mathbf{x}) + \varepsilon$ with $Y \in \Re$ and $\varepsilon \approx N(0, \sigma^2)$, based on a given sample of data $D = \{\langle x_i^1, \cdots, x_i^p, y_i \rangle\}_{i=1}^n$. The obtained models consist of a hierarchy of logical tests on the values of any of the p predictor variables. The terminal nodes of these trees, known as the leaves, contain the numerical predictions of the model for the target variable Y.

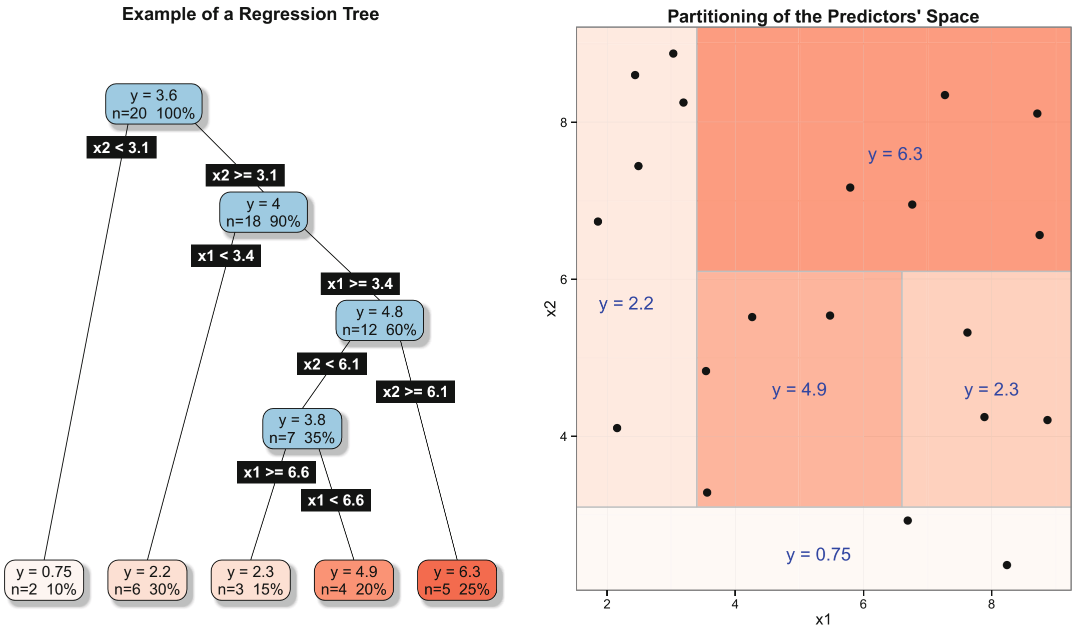
Motivation and Background

Work on regression trees goes back to the AID system by Morgan and Sonquist (1963). Nonetheless, the seminal work is the book *Classification and Regression Trees* by Breiman and colleagues (1984). This book has established several standards in many theoretical aspects of tree-based regression, including over-fitting avoidance by post-pruning, the notion of surrogate splits for handling unknown variable, and estimating variable importance.

Regression trees have several features that make them a very interesting approach to several multiple regression problems. Namely, regression trees provide (i) automatic variable selection making them highly insensitive to irrelevant variables, (ii) computational efficiency that allows addressing large problems, (iii) handling of unknown variable values, (iv) handling of both numerical and nominal predictor variables, (v) insensitivity to predictors' scales, and (vi) interpretable models for most domains. In spite of all these advantages, regression trees have poor prediction accuracy in several domains because of the piecewise constant approximation they provide, and they are also unstable with respect to small changes on the training data.

Structure of Learning System

The most common regression trees are binary with logical tests in each node (an example is given on the left graph of Fig. 1). Tests on numerical variables usually take the form $x_i < \alpha$, with $\alpha \in \Re$, while tests on nominal variables have the form $x_j \in \{v_1, \cdots, v_m\}$. Each path from the root (top) node to a leaf can be seen as a logical assertion defining a region on the predictors' space. Any regression tree provides a full mutually exclusive partition of the predictor space into L regions with boundaries that are



Regression Trees, Fig. 1 A regression tree and the partitioning it provides

parallel to the predictors' axes due to the form of the tests. Figure 1 illustrates these ideas with a tree and the respective partitioning on the right side of the graph.

Using a regression tree for obtaining predictions for new observations is straightforward. For each new observation, we follow a path from the root node to a leaf selecting the branches according to the variable values of the observation. All observations falling in a partition are predicted with the same constant value, and that is the reason for regression trees sometimes being referred to as piecewise constant models. In effect, the approximation provided by a regression tree is given by

$$Y = \sum_{l \in \mathcal{L}} k_l \cdot I(P_l) \qquad (1)$$

where $\mathcal{L}$ is the set of L leaves, the ks are the constants at each leaf, $I()$ is an indicator function, and P_i is a logical assertion formed by the conjunction of conditions from the root node till the leaf i. For instance, the rightmost leaf of the tree in Fig. 1 is described by the logical assertion $x_2 \geq 3.1 \wedge x_1 \geq 3.4 \wedge x_2 \geq 6.1$, which is equivalent to $x_1 \geq 3.4 \wedge x_2 \geq 6.1$.

Learning a Regression Tree

A binary regression tree is obtained by a very efficient algorithm known as recursive partitioning (Algorithm 1).

If the termination criterion is not met by the input sample D, the algorithm selects the best logical test on one of the predictor variables according to some criterion. This test divides the current sample in two partitions: the one with the cases satisfying the test and the remaining. The algorithm proceeds by recursively applying the same method to these two partitions to obtain the left and right branches of the node. Algorithm 1 has three main components that characterize the type of regression tree we are obtaining: (i) the *termination criterion*, (ii) the *constant* k, and (iii) the method to find the *best test on one of the predictors*. The choices for these components are related to the preference criteria that are used to build the trees. The most common criterion is the minimization of the sum of the square errors, known as the least squares (LS) criterion. Using this criterion it can be easily proven (e.g., Breiman et al. 1984) that the constant k should be the average target variable value of the cases in the leaf. With respect to

Algorithm 1 Recursive partitioning

```
function RECURSIVEPARTITIONING($D$)
  $Input$ :    $D$, a sample of cases, $\{\langle x_i^1, \cdots, x_i^p, y_i \rangle\}$
  $Output$ :    $t$, a tree node

    if <TERMINATION CRITERION> then
        Return a leaf node with <CONSTANT K>
    else
        $t \leftarrow$ new tree node
        $t.split \leftarrow$ <FIND THE BEST TEST ON ONE OF THE VARIABLES>
        $t.leftNode \leftarrow$ RecursivePartitioning($\{\mathbf{x} \in D : \mathbf{x} \vDash t.split\}$)
        $t.rightNode \leftarrow$ RecursivePartitioning($\{\mathbf{x} \in D : \mathbf{x} \nvDash t.split\}$)
        Return the node $t$
    end if
end function
```

the *termination criterion*, usually very relaxed settings are selected so that an overly large tree is grown. The reasoning is that the trees will be pruned afterward with the goal of overcoming the problem of over-fitting of the training data.

According to the LS criterion, the error in a given node is given by

$$Err(t) = \frac{1}{n_t} \sum_{\langle \mathbf{x}_i, y_i \rangle \in D_t} (y_i - k_t)^2 \qquad (2)$$

where D_t is the sample of cases in node t, n_t is the cardinality of this set, and k_t is the average target variable value of the cases in D_t.

Any logical test s divides the cases in D_t in two partitions, D_{t_L} and D_{t_R}. The resulting pooled error is given by

$$Err(t,s) = \frac{n_{t_L}}{n_t} \times Err(t_L) + \frac{n_{t_R}}{n_t} \times Err(t_R) \qquad (3)$$

where n_{t_L}/n_t (n_{t_R}/n_t) is the proportion of cases going to the left (right) branch of t.

In this context, we can estimate the value of the split s by the respective error reduction, and this can be used to evaluate all candidate splits test for a node:

$$\Delta(s,t) = Err(t) - Err(t,s) \qquad (4)$$

Finding the best split test for a node t involves evaluating all possible tests for this node using Eq. 4. For each predictor of the problem, one needs to evaluate all possible splits in that variable. For continuous variables, this requires a sorting operation on the values of this variable occurring in the node. After this sorting, a fast incremental algorithm can be used to find the best cut point value for the test (e.g., Torgo 1999). With respect to nominal variables, Breiman and colleagues (1984) have proved a theorem that avoids trying all possible combinations of values, reducing the computational complexity of this task from $O(2^{v-1} - 1)$ to $O(v - 1)$, where v is the number of values of the nominal variable.

Departures from the standard learning procedure described above include, among others, the use of multivariate split nodes (e.g., Breiman et al. 1984, Li et al. 2000, and Gama 2004) to overcome the axis parallel representation limitation of partitions, the use of different criteria to find the best split node (e.g., Robnik-Sikonja and Kononenko 1996, Buja and Lee 2001, and Loh 2002), the use of different preference criteria to guide the tree growth (e.g., Breiman et al. 1984, Torgo 1999, Buja and Lee 2001, and Torgo and Ribeiro 2003), and the use of both regression and split nodes (e.g., Lubinsky 1995 and Malerba et al. 2004).

Pruning Regression Trees

As most nonparametric modeling techniques, regression trees may over-fit the training data which will inevitably lead to poor out of the sample predictive performance. The standard procedure to fight this undesirable effect is to grow an overly large tree and then to use some reliable error estimation procedure to find the "best" sub-tree of this large model. This procedure is known as post-pruning a tree (Breiman et al. 1984). An alternative is to stop tree growth sooner in a process known as pre-pruning which again needs to be guided by reliable error estimation to known when over-fitting is starting to occur. Although more efficient in computational terms, this latter alternative may lead to stop tree growth too soon even with look-ahead mechanisms.

Post-pruning is usually carried out in a three-stage procedure: (i) a set of sub-trees of the initial tree is generated, (ii) some reliable error estimation procedure is used to obtain estimates for each member of this set, and (iii) some method based on these estimates is used to select one of these trees as the final tree model. Different methods exist for each of these steps. A common setup (e.g., Breiman et al. 1984) is to use error-complexity pruning to generate a sequence of nested sub-trees, whose error is then estimated by cross validation. The final tree is selected using the x-SE rule which starts with the lowest estimated error sub-tree and then selects the smallest tree within the interval of x standard errors of the lowest estimated error tree (a frequent setting is to use 1 standard error).

Variations on the subject of pruning regression trees include, among others, pre-pruning alternatives (e.g., Breiman and Meisel 1976 and Friedman 1979), the use of different tree error estimators (see Torgo (1998) for a comparative study and references to different alternatives), and the use of the MDL principle to guide the pruning (Robnik-Sikonja and Kononenko 1998).

Cross-References

- ▶ Model Trees
- ▶ Random Forests
- ▶ Regression
- ▶ Supervised Learning

Recommended Reading

Breiman L, Friedman J, Olshen R, Stone C (1984) Classification and regression trees. Statistics/probability series. Wadsworth & Brooks/Cole Advanced Books & Software, Belmont

Breiman L, Meisel WS (1976) General estimates of the intrinsic variability of data in nonlinear regression models. J Am Stat Assoc 71:301–307

Buja A, Lee Y-S (2001) Data mining criteria for tree-based regression and classification. In: Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining, San Francisco, pp 27–36

Friedman JH (1979) A tree-structured approach to nonparametric multiple regression. In: Gasser T, Rosenblatt M (eds) Smoothing techniques for curve estimation. Lecture notes in mathematics, vol 757. Springer, Berlin/New York, pp 5–22

Gama J (2004) Functional trees. Mach Learn 55(3):219–250

Li KC, Lue H, Chen C (2000) Interactive tree-structured regression via principal Hessians direction. J Am Stat Assoc 95:547–560

Loh W (2002) Regression trees with unbiased variable selection and interaction detection. Stat Sin 12: 361–386

Lubinsky D (1995) Tree structured interpretable regression. In: Proceedings of the workshop on AI & statistics, Key West

Malerba D, Esposito F, Ceci M, Appice A (2004) Top-down induction of model trees with regression and splitting nodes. IEEE Trans Pattern Anal Mach Intell 26(5):612–625

Morgan JN, Sonquist JA (1963) Problems in the analysis of survey data, and a proposal. J Am Stat Assoc 58(302):415–434

Robnik-Sikonja M, Kononenko I (1996) Context-sensitive attribute estimation in regression. In: Proceedings of the ICML-96 workshop on learning in context-sensitive domains, Bari

Robnik-Sikonja M, Kononenko I (1998) Pruning regression trees with MDL. In: Proceedings of ECAI-98, Brighton

Torgo L (1998) Error estimates for pruning regression trees. In: Nedellec C, Rouveirol C (eds) Proceedings of the 10th European conference on machine learning, Chemnitz. LNAI, vol 1398. Springer

Torgo L (1999) Inductive learning of tree-based regression models. PhD thesis, Faculty of Sciences, Department of Computer Science, University of Porto

Torgo L, Ribeiro R (2003) Predicting outliers. In: Lavrac N, Gamberger D, Todorovski L, Blockeel H (eds) Proceedings of principles of data mining and knowledge discovery (PKDD'03), Cavtat/Dubronik. LNAI, vol 2838. Springer, pp 447–458

Regularization

Xinhua Zhang
NICTA, Australian National University,
Canberra, ACT, Australia
School of Computer Science, Australian
National University, Canberra, ACT, Australia
NICTA London Circuit, Canberra, ACT,
Australia

Abstract

Regularization plays a key role in many machine learning algorithms. Exactly fitting a model to the training data is generally undesirable, because it will fit the noise in the training examples (▸ overfitting), and is doomed to predict (generalize) poorly on unseen data. In contrast, a simple model that fits the training data well is more likely to capture the regularities in it and generalize well. A number of regularizers have been proposed for various applications, and theoretical tools that characterize their complexity are also available.

Definition

In general, a regularizer a quantifier of the complexity of a model, and many successful machine learning algorithms fall in the framework of regularized risk minimization:

$$\text{(How well the model fits the training data)} \tag{1}$$

$$+\lambda \cdot \text{(complexity/regularization of the model)}, \tag{2}$$

where the positive real number λ controls the trade-off.

There is a variety of regularizers, which yield different statistical and computational properties. In general, there is no universally best regularizer, and a regularization approach must be chosen depending on the dataset.

Motivation and Background

The main goal of machine learning is to induce a model from the observed data and use this model to make predictions and decisions. This is also largely the goal of general natural science and is commonly called inverse problems ("forward problem" means using the model to generate observations). Therefore, it is no surprise that regularization had been well studied before the emergence of machine learning.

Inverse problems are typically ill posed, e.g., having only a finite number of samples drawn from an uncountable space or having a finite number of measurements in an infinite dimensional space. In machine learning, we often need to induce a classifier for the whole feature-label space, while only a finite number of feature-label pairs are available for training. In practice, the set of candidate models is often flexible enough to precisely fit all the training examples. However, this can lead to significant overfitting when the training data is noisy, and the real challenge is how to generalize well on the unseen data in the whole feature-label space.

Many techniques have been proposed to tackle ill-posed inverse problems. Almost all of them introduce an additional measure on how much a model is preferred a priori (i.e., without observing the training data). This extra belief on the desirable form of the model reflects the external knowledge of the model designer. It cannot be replaced by the data itself according to the "no free lunch theorem," which states that if there is no assumption on the mechanism of labeling, then it is impossible to generalize, and any model can be inferior to another on some distribution of the feature-label pair (Devroye et al. 1996).

A commonly used prior is the so-called ▸ Occam's razor, which prefers "simple" models. It asserts that among all the models which fit the training data well, the simplest one is more likely to capture the "regularities" in it and hence has a larger chance to generalize well to the unseen data. Then an immediate question is how to quantify the complexity of a model, which is often called a regularizer. Intuitively, a regularizer can encode preference for a sparse model (few features are relevant for prediction), a large margin model (two classes have a wide margin), or a smooth model with weak high-frequency components. A general framework of regularization was given by Tikhonov (1943).

Theory

Suppose n feature-label pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ are drawn *iid* from a certain joint distribution P on $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are the spaces of feature and label, respectively. Let the marginal distribution on $\mathcal{X}$ and $\mathcal{Y}$ be P_x and P_y respectively. For convenience, let $\mathcal{X}$ be $\mathbb{R}^p$ (Euclidean space). Denote $X := (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and $\mathbf{y} := (y_1, \ldots, y_n)^\top$.

An Illustrative Example: Ridge Regression

Ridge regression is illustrative of the use of regularization. It tries to fit the label y by a linear model $\langle \mathbf{w}, \mathbf{x} \rangle$ (inner product). So we need to solve a system of linear equations in $\mathbf{w}$: $(\mathbf{x}_1, \ldots, \mathbf{x}_n)^\top \mathbf{w} = \mathbf{y}$, which is equivalent to a linear least square problem: $\min_{\mathbf{w} \in \mathbb{R}^p} \left\| X^\top \mathbf{w} - \mathbf{y} \right\|^2$. If the rank of X is less than the dimension of $\mathbf{w}$, then it is overdetermined and the solution is not unique.

To approach this ill-posed problem, one needs to introduce additional assumptions on what models are preferred, i.e., the regularizer. One choice is to pick a matrix Γ and regularize $\mathbf{w}$ by $\|\Gamma \mathbf{w}\|^2$. As a result we solve $\min_{\mathbf{w} \in \mathbb{R}^p} \left\| X^\top \mathbf{w} - \mathbf{y} \right\|^2 + \lambda \left\| \Gamma^\top \mathbf{w} \right\|^2$, and the solution has a closed form $\mathbf{w}^* = (XX^\top + \lambda \Gamma \Gamma^\top) X \mathbf{y}$. Γ can be simply the identity matrix which encodes our preference for small norm models.

The use of regularization can also be justified from a Bayesian point of view. Treating $\mathbf{w}$ as a multivariate random variable and the likelihood as $\exp\left(-\left\| X^\top \mathbf{w} - \mathbf{y} \right\|^2\right)$, then the minimizer of $\left\| X^\top \mathbf{w} - \mathbf{y} \right\|^2$ is just a maximum likelihood estimate of $\mathbf{w}$. However, we may also assume a prior distribution over $\mathbf{w}$, e.g., a Gaussian prior $p(\mathbf{w}) \sim \exp\left(-\lambda \left\| \Gamma^\top \mathbf{w} \right\|^2\right)$. Then the solution of the ridge regression is simply the maximum a posterior estimate of $\mathbf{w}$.

Examples of Regularization

A common approach to regularization is to penalize a model by its complexity measured by some real-valued function, e.g., a certain "norm" of $\mathbf{w}$. We list some examples below.

***L_1* regularization** L_1 regularizer, $\|\mathbf{w}\|_1 := \sum_i |w_i|$, is a popular approach to finding sparse models, i.e., only a few components of $\mathbf{w}$ are nonzero, and only a corresponding small number of features are relevant to the prediction. A well-known example is the LASSO algorithm (Tibshirani 1996), which uses a L_1-regularized least square:

$$\min_{\mathbf{w} \in \mathbb{R}^p} \left\| X^\top \mathbf{w} - \mathbf{y} \right\|^2 + \lambda \|\mathbf{w}\|_1 .$$

***L_2* regularization** The L_2 regularizer, $\|\mathbf{w}\|_2 := \sqrt{\sum_i |w_i|^2}$, is popular due to its self-dual properties. In all L_p spaces, only the L_2 space is Hilbertian and self-adjoint, so it affords much convenience in studying and exploiting the dual properties of the L_2-regularized models. A well-known example is the support vector machines (SVMs), which minimize the L_2-regularized hinge loss:

$$\frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - \langle \mathbf{w}, \mathbf{x}_i \rangle\} + \lambda \|\mathbf{w}\|_2^2 .$$

***L_p* regularization** In general, all L_p norms $\|\mathbf{w}\|_p := \left(\sum_i |w_i|^p\right)^{1/p}$ $(p \geq 1)$ can be used for regularization. When $p < 1$, $\left(\sum_i |w_i|^p\right)^{1/p}$ is no longer convex. A specially interesting case is when $p = 0$, and $\|\mathbf{w}\|_0$ is defined as the number of nonzero elements in $\mathbf{w}$ (the sparseness of $\mathbf{w}$). But explicitly optimizing the L_0 norm leads to a combinatorial problem which is hard to solve. In some cases, the L_1 regularizer can approximately recover the solution of L_0 regularization (Candes and Tao 2005).

***$L_{p,q}$* regularizer** The $L_{p,q}$ regularizer is popular in the context of multitask learning (Tropp 2006). Suppose there are T tasks, and each training example $\mathbf{x}_i$ has a label vector $\mathbf{y}_i \in \mathbb{R}^T$ with each component corresponding to a task.

For each task t, we seek for a linear regressor $\langle \mathbf{w}_t, \mathbf{x} \rangle$ such that for each training example $\mathbf{x}_i$, $\langle \mathbf{w}_t, \mathbf{x}_i \rangle$ fits the t-th component of $\mathbf{y}_i$. Of course, the $\mathbf{w}_t$ could be determined independently from each other. But in many applications, the T tasks are somehow related, and it will be advantageous to learn them as a whole. Stack $\mathbf{w}_t$'s into a matrix $W := (\mathbf{w}_1, \ldots, \mathbf{w}_T)$ where each column corresponds to a task and each row corresponds to a feature. Then the intuition of multitask learning can be concretized by regularizing W with the $L_{p,q}$ compositional norm ($p, q \geq 1$):

$$\|W\|_{p,q} := \left(\sum_i \left(\sum_t |w_{it}|^p \right)^{\frac{q}{p}} \right)^{\frac{1}{q}},$$

where w_{it} is the i-th component of $\mathbf{w}_t$. When $q = 1$, it becomes the L_1 norm of the L_p norm of the rows, and the sparse inducing property of L_1 norm encourages the rows to have L_p norm 0, i.e., the corresponding feature is not used by *any* task. Other choices of p and q are also possible.

Entropy regularizer The entropy regularizer is useful in boosting, and it works in a slightly different way from the above regularizers. Boosting aims to find a convex combination of hypotheses, such that the training data is accurately classified by the ensemble. At each step, the boosting algorithm maintains a distribution $\mathbf{d}$ ($d_i > 0$ and $\sum_i d_i = 1$) over the training examples, feeds $\mathbf{d}$ to an oracle which returns a new hypothesis, and then updates $\mathbf{d}$ and go on. As a "simple" ensemble means a small number of weak hypotheses, the boosting algorithm is expected to find an accurate ensemble by taking as few steps as possible. This can be achieved by exponentiated gradient descent (Kivinen and Warmuth 1997), which stems from the relative entropy regularizer $\sum_i d_i \log \frac{d_i}{1/n}$ applied at each step. It also attracts $\mathbf{d}$ toward the uniform distribution, which helps avoid overfitting the noise, i.e., trying hard to match the (incorrect) label of a few training examples.

Miscellaneous Instead of using a function that directly measures the complexity of the model $\mathbf{w}$, regularization can also be achieved by penalizing the complexity of the *output* of the model on the training data. This is called value regularization (Rifkin and Lippert 2007). It not only yields neat derivations of standard algorithms but also provides much convenience in studying the learning theory and optimization.

Furthermore, the regularized risk minimization framework in (1) is not the only approach to regularization. For example, in online learning where the model is updated iteratively, early stopping is an effective form of regularization, and it has been widely used in training neural networks. Suppose the available dataset is divided into a training set and a validation set and the model is learned online from the training set, then the algorithm terminates when the performance of the model on the validation set stops improving.

Measuring the Capacity of Model Class

Besides penalizing the complexity of the model, we can restrict the complexity of the model class $\mathcal{F}$ in the first place. For example linear regression is intrinsically "simpler" than quadratic regression. Decision stumps are "simpler" than linear classifiers. In other words, regularization can be achieved by restricting the capacity of the model class, and the key question is how to quantify this capacity. Some commonly used measures in the context of binary classification are the following:

VC dimension The Vapnik-Chervonenkis dimension (▶ VC dimension) quantifies how many data points can be arbitrarily labeled by using the functions in $\mathcal{F}$ (Vapnik and Chervonenkis 1971). $\mathcal{F}$ is said to *shatter* a set of data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ if, for any assignment of labels to these points, there exists a function $f \in \mathcal{F}$ which yields this labeling. The VC dimension of $\mathcal{F}$ is the maximum n such that any n data points can be shattered by $\mathcal{F}$. For example, decision stumps have VC dimension 2, and linear classifiers (with bias) in a p dimensional space have VC dimension $p + 1$.

Covering number The idea of covering number (Guo et al. 1999) is to characterize the inherent "dimension" of $\mathcal{F}$, in a way that follows the standard concept of vector dimension. Given n data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, we may endow the model class $\mathcal{F}$ with the following metric:

$$d_n(f, g) := \frac{1}{n}\sum_{i=1}^{n} \delta(f(\mathbf{x}_i) \neq g(\mathbf{x}_i)), \ \forall\, f, g \in \mathcal{F},$$

where $\delta(\cdot) = 1$ if $\cdot$ is true and 0 otherwise. A set of functions $f_1, \ldots, f_m$ is said to be a cover of $\mathcal{F}$ at radius ϵ if, for any function $f \in \mathcal{F}$, there exists an f_i such that $d_n(f, f_i) < \epsilon$. Then the covering number of $\mathcal{F}$ at radius $\epsilon > 0$ with respect to d_n is the minimum size of a cover of radius ϵ.

To understand the motivation of the definition, consider the unit ball in $\mathbb{R}^p$. To cover it by ϵ radius balls, one needs order $N(\epsilon, p) = \epsilon^{-p}$ balls. Then the dimension p can be estimated from the rate of growth of $\log N(\epsilon, p) = -p \log \epsilon$ with respect to ϵ. The covering number is an analogy of $N(\epsilon, p)$, and the dimension of $\mathcal{F}$ can be estimated in the same spirit.

Rademacher average The Rademacher average is a soft variant of the VC dimension. Instead of requiring the model class to shatter n data points, it allows that the labels be violated at some cost. Let $\sigma_i \in \{-1, 1\}$ be an arbitrary assignment of the labels, and assume all functions in $\mathcal{F}$ range in $\{-1, 1\}$ (this restriction can be relaxed). Then a model $f \in \mathcal{F}$ is considered as the most consistent with $\{\sigma_i\}$ if it maximizes $\frac{1}{n}\sum_{i=1}^{n} \sigma_i f(\mathbf{x}_i)$. This term equals 1 if $\mathcal{F}$ does contain a model consistent with $\{\sigma_i\}$. Then we take an average over all possible assignments of σ_i, i.e., treating σ_i as a binary random variable with $P(\sigma_i = 1) = P(\sigma_i = -1) = 0.5$ and calculating the expectation over $\{\sigma_i\}$:

$$\mathcal{R}_n(\mathcal{F}) = \mathop{\mathbb{E}}_{\sigma}\left[\sup_{f \in \mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \sigma_i f(\mathbf{x}_i)\right].$$

Furthermore, we may take expectation over the samples $\mathbf{x}_1, \ldots, \mathbf{x}_n$:

$$\mathcal{R}(\mathcal{F}) = \mathop{\mathbb{E}}_{\mathbf{x}_i \sim P_x} \mathop{\mathbb{E}}_{\sigma}\left[\sup_{f \in \mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \sigma_i f(\mathbf{x}_i)\right].$$

Therefore, similar to VC dimension, the Rademacher average is high if the model class $\mathcal{F}$ is "rich" and can match most assignments of $\{\sigma_i\}$.

Applications

In many applications such as bioinformatics, the training examples are expensive and the number of features p is much higher than the number of labeled examples n. In such cases, regularization is crucial, e.g., Zhang et al. (2008).

L_1 regularization has gained much popularity recently in the field of compressed sensing, and it has been widely used in imaging for radar, astronomy, medical diagnosis, and geophysics. See an ensemble of publications at http://dsp.rice.edu/cs.

The main spirit of regularization, namely, a preference for models with lower complexity, has been used by some ▸ model evaluation techniques. Examples include Akaike information criterion (AIC), Bayesian information criterion (BIC), ▸ minimum description length (MDL), and the minimum message length (MML).

Cross-References

▸ Minimum Description Length Principle
▸ Model Evaluation
▸ Occam's Razor
▸ Overfitting
▸ Support Vector Machines
▸ VC Dimension

Recommended Reading

Regularization lies at the heart of statistical machine learning, and it is indispensable in almost every learning algorithm. A comprehensive statistical analysis from the computational learning theory perspective can be found in Bousquet et al. (2005) and Vapnik (1998). Abundant resources on compressed sensing including both theory and applications are available at http://dsp.rice.edu/cs. Regularizations related to SVMs and kernel methods are discussed in detail by Schölkopf and Smola (2002) and Shawe-Taylor and Cristianini (2004). Anthony and Bartlett (1999) provide in-depth theoretical analysis for neural networks.

Anthony M, Bartlett PL (1999) Neural network learning: theoretical foundations. Cambridge University Press, Cambridge

Bousquet O, Boucheron S, Lugosi G (2005) Theory of classification: a survey of recent advances. ESAIM: Probab Stat 9:323–375

Candes E, Tao T (2005) Decoding by linear programming. IEEE Trans Inf Theory 51(12): 4203–4215

Devroye L, Györfi L, Lugosi G (1996) A probabilistic theory of pattern recognition. Applications of mathematics, vol 31. Springer, New York

Guo Y, Bartlett PL, Shawe-Taylor J, Williamson RC (1999) Covering numbers for support vector machines. In: Proceedings annual conference computational learning theory. Montreal, Canada

Kivinen J, Warmuth MK (1997) Exponentiated gradient versus gradient descent for linear predictors. Inf Comput 132(1):1–64

Rifkin RM, Lippert RA (2007) Value regularization and Fenchel duality. J Mach Learn Res 8: 441–479

Schölkopf B, Smola A (2002) Learning with kernels. MIT Press, Cambridge

Shawe-Taylor J, Cristianini N (2004) Kernel methods for pattern analysis. Cambridge University Press, Cambridge

Tibshirani R (1996) Regression shrinkage and selection via the LASSO. J R Stat Soc Ser B Stat Methodol 58:267–288

Tikhonov AN (1943) On the stability of inverse problems. Dokl Akad Nauk SSSR 39(5):195–198

Tropp JA (2006) Algorithms for simultaneous sparse approximation, Part II: convex relaxation. Signal Process 86(3):589C–602

Vapnik V (1998) Statistical learning theory. Wiley, New York

Vapnik V, Chervonenkis A (1971) On the uniform convergence of relative frequencies of events to their probabilities. Theory Probab Appl 16(2): 264–281

Zhang M, Zhang D, Wells MT (2008) Variable selection for large *p* small *n* regression models with incomplete data: mapping Qtl with epistases. BMC Bioinf 9:251

Regularization Networks

► Radial Basis Function Networks

Reinforcement Learning

Peter Stone
Department of Computer Science, The
University of Texas at Austin, Austin, TX, USA

Abstract

This entry provides an overview of Reinforcement Learning (RL), with cross-references to specific RL algorithms.

Reinforcement learning describes a large class of learning problems characteristic of autonomous agents interacting in an environment: sequential decision-making problems with delayed reward. Reinforcement-learning algorithms seek to learn a policy (mapping from states to actions) that maximizes the reward received over time.

Unlike in ► supervised learning problems, in reinforcement-learning problems, there are no labeled examples of correct and incorrect behavior. However, unlike ► unsupervised learning problems, a reward signal can be perceived.

Many different algorithms for solving reinforcement-learning problems are covered in other entries. This entry provides just a brief high-level classification of the algorithms.

Perhaps the most well-known approach to solving reinforcement-learning problems, as covered in detail by Sutton and Barto (1998), is based on learning a value function, which represents the long-term expected reward of each state the agent may encounter, given a particular policy. This approach typically assumes that the environment is a ► Markov decision process in which rewards are discounted over time, though it is also possible to optimize for average reward per time step as in ► average-reward reinforcement learning. If a complete model of the environment is available, ► dynamic programming, or specifically ► value iteration, can be used to compute an optimal value function, from which an optimal policy can be derived.

If a model is not available, an optimal value function can be learned from experience via model-free techniques such as ▸ temporal difference learning, which combine elements of dynamic programming with Monte Carlo estimation. Partly due to Watkins' elegant proof that ▸ Q-learning converges to the optimal value function (Watkins 1989), temporal difference methods are currently among the most commonly used approaches for reinforcement-learning problems.

Watkins' convergence proof relies on executing a policy that visits every state infinitely often. In practice, Q-learning does converge in small, discrete domains. However in larger and particularly in continuous domains, the learning algorithm must generalize the value function across states, a process known as ▸ value function approximation. Examples include ▸ instance-based reinforcement learning, ▸ Gaussian process reinforcement learning, and ▸ relational reinforcement learning.

Even when combined with value function approximation, the most basic value-free methods, such as Q-learning and SARSA, are very inefficient with respect to experience: they are not sample-efficient. With the view that experience is often more costly than computation, much research has been devoted to making more efficient use of experience, for instance, via ▸ hierarchical reinforcement learning, ▸ reward shaping, or ▸ model-based reinforcement learning in which the experience is used to learn a domain model, which can then be solved via dynamic programming.

Though these methods make efficient use of the experience that is presented to them, the goal of optimizing sample efficiency also motivates the study of ▸ efficient exploration in reinforcement learning. The study of exploration methods can be isolated from the full reinforcement-learning problem by removing the notion of temporally delayed reward as is done in ▸ associative reinforcement learning or by removing the notion of states altogether as is done in ▸ k-armed bandits. k-Armed bandit algorithms focus entirely on the exploration versus exploitation challenge, without having to worry about generalization across states or delayed rewards. Back in the context of the full RL problem, ▸ Bayesian reinforcement learning enables optimal exploration given prior distributions over the parameters of the learning problem. However, its computational complexity has limited its use so far to very small domains.

Although most of the methods above revolve around learning a value function, reinforcement-learning problems can also be solved without learning value functions, by directly searching the space of potential policies via policy search. Effective ways of conducting such a search include ▸ policy gradient reinforcement learning, ▸ least squares reinforcement-learning methods, and evolutionary reinforcement learning.

As typically formulated, the goal of a reinforcement-learning algorithm is to learn an optimal (or high-performing) policy based on knowledge of, or experience of, a reward function (and state transition function). However, it is also possible to take the opposite perspective that of trying to learn the reward function based on observation of the optimal policy. This problem formulation is known as ▸ inverse reinforcement learning.

Leveraging this large body of theory and algorithms, a current focus in the field is deploying large-scale, successful applications of reinforcement learning. Two such applications treated herein are ▸ autonomous helicopter flight using reinforcement learning and ▸ robot learning.

Cross-References

- ▸ Associative Reinforcement Learning
- ▸ Autonomous Helicopter Flight Using Reinforcement Learning
- ▸ Average-Reward Reinforcement Learning
- ▸ Bayesian Reinforcement Learning
- ▸ Dynamic Programming
- ▸ Efficient Exploration in Reinforcement Learning
- ▸ Gaussian Process Reinforcement Learning
- ▸ Hierarchical Reinforcement Learning

- ▸ Instance-Based Reinforcement Learning
- ▸ Inverse Reinforcement Learning
- ▸ Least-Squares Reinforcement Learning Methods
- ▸ Model-Based Reinforcement Learning
- ▸ Policy Gradient Methods
- ▸ Q-Learning
- ▸ Relational Reinforcement Learning
- ▸ Reward Shaping
- ▸ Symbolic Dynamic Programming
- ▸ Temporal Difference Learning
- ▸ Value Function Approximation

Recommended Reading

Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT, Cambridge

Watkins CJCH (1989) Learning from delayed rewards. PhD thesis, King's College, Cambridge

Reinforcement Learning in Structured Domains

▸ Relational Reinforcement Learning

Relational Data Mining

▸ Inductive Logic Programming

Relational Dynamic Programming

▸ Symbolic Dynamic Programming

Relational Learning

Jan Struyf[1] and Hendrik Blockeel[1,2]
[1]Katholieke Universiteit Leuven, Leuven,
Heverlee, Leuven, Belgium
[2]Leiden Institute of Advanced Computer
Science, Heverlee, Belgium

Problem Definition

Relational learning refers to learning in a context where there may be relationships between learning examples, or where these examples may have a complex internal structure (i.e., consist of multiple components and there may be relationships between these components). In other words, the "relational" may refer to both an internal or external relational structure describing the examples. In fact, there is no essential difference between these two cases, as it depends on the definition of an example whether relations are internal or external to it. Most methods, however, are clearly set in one of these two contexts.

Learning from Examples with External Relationships

This setting considers learning from a set of examples where each example itself has a relatively simple description, for instance in the attribute-value format, and relationships may be present among these examples.

Example 1 Consider the task of web-page classification. Each web-page is described by a fixed set of attributes, such as a bag of words representation of the page. Web-pages may be related through hyperlinks, and the class label of a given page typically depends on the labels of the pages to which it links.

Example 2 Consider the Internet Movie Database (www.imdb.com). Each movie is described by a fixed set of attributes, such as its title and genre. Movies are related to other entity types, such as *Studio*, *Director*, *Producer*, and *Actor*, each of which is in turn described by a different set of attributes. Note that two movies can be related through the other entity types. For example, they can be made by the same studio or star the same well-known actor. The learning task in this domain could be, for instance, predicting the opening weekend box office receipts of the movies.

If relationships are present among examples, then the examples may not be independent and identically distributed (i.i.d.), an assumption made by many learning algorithms. Relational data that violates this assumption can be detrimental to learning performance as Jensen and Neville (2002) show. Relationships among examples can, on the other hand, also be exploited by the learning algorithm. ▶ Collective classification techniques (Jensen et al. 2004), for example, take the class labels of related examples into account when classifying a new instance.

Learning from Examples with a Complex Internal Structure

In this setting, each example may have a complex internal structure, but no relationships exist that relate different examples to one another. Learning algorithms typically use individual-centered representations in this setting, such as logical interpretations or strongly typed terms (Lloyd 2003), which store together all data of a given example. An important advantage of individual-centered representations is that they scale better to large datasets. Special cases of this setting include applications where the examples can be represented as graphs, trees, or sequences.

Example 3 Consider a database of candidate chemical compounds to be used in drugs. The molecular structure of each compound can be represented as a graph where the vertices are atoms and the edges are bonds. Each atom is labeled with its element type and the bonds can be single, double, triple, or aromatic bonds. Compounds are classified as active or inactive with regard to a given disease and the goal is to build models that are able to distinguish active from inactive compounds based on their molecular structure. Such models can, for instance, be used to gain insight in the common substructures, such as binding sites, that determine a compound's activity.

Approaches to Relational Learning

Many different kinds of learning tasks have been defined in relational learning, and an even larger number of approaches have been proposed for tackling these tasks. We give an overview of different learning settings that can be considered instances of relational learning.

Inductive Logic Programming

In ▶ inductive logic programming (ILP), the input and output knowledge of a learner are described in (variants of) first-order predicate logic. Languages based on first-order logic are highly expressive from the point of view of knowledge representation, and indeed, a language such as Prolog (Bratko 1986) can be used without adaptations to represent objects and the relationships between them, as well as background knowledge that one may have about the domain.

Example 4 This example is based on the work by Finn et al. (1998). Consider a data set that describes chemical compounds. The active compounds in the set are ACE inhibitors, which are used in treatments for hypertension. The molecular structure of the compounds is represented as a set of Prolog facts, such as: atom(m1, a1, o).

atom(m1, a2, c).
...
bond(m1, a1, a2, 1).
...
coord(m1, a1, 5.91, – 2.44, 1.79).
coord(m1, a2, 0.57, – 2.77, 0.33).
...

which states that molecule m1 includes an oxygen atom a1 and a carbon atom a2 that are single bonded. The coord/5 predicate lists the 3D coordinates of the atoms in the given conformer. Background knowledge, such as the concepts zinc site, hydrogen donor, and the distance between atoms, are defined by means of Prolog clauses. Figure 1 shows a clause learned by the inductive logic programming system Progol (Džeroski and Lavraè 2001, Ch. 7) that makes use of these background knowledge predicates. This clause is the description of a pharmacophore, that is, a submolecular structure that causes a certain observable property of a molecule.

More details on the theory of inductive logic programming and descriptions of algorithms can be found in the entry on ▶ Inductive Logic Programming in this encyclopedia, or in references (De Raedt 2008; Džeroski and Lavraè 2001).

R

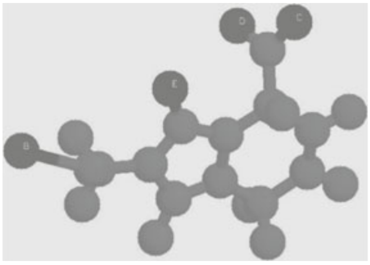
a

```
ACE_inhibitor(A) :-
     zincsite(A, B),
     hacc(A, C),
     dist(A, B, C, 7.9, 1.0),
     hacc(A, D),
     dist(A, B, D, 8.5, 1.0),
     dist(A, C, D, 2.1, 1.0),
     hacc(A, E),
     dist(A, B, E, 4.9, 1.0),
     dist(A, C, E, 3.1, 1.0),
     dist(A, D, E, 3.8, 1.0).
```

b

Molecule A is an ACE inhibitor if:
molecule A can bind to zinc at site B, and
molecule A contains a hydrogen acceptor C, and
the distance between B and C is 7.9 ± 1.0Å, and
molecule A contains a hydrogen acceptor D, and
the distance between B and D is 8.5 ± 1.0Å, and
the distance between C and D is 2.1 ± 1.0Å, and
molecule A contains a hydrogen acceptor E, and
the distance between B and E is 4.9 ± 1.0Å, and
the distance between C and E is 3.1 ± 1.0Å, and
the distance between D and E is 3.8 ± 1.0Å.

c



Relational Learning, Fig. 1 (**a**) Prolog clause modeling the concept of an ACE inhibitor in terms of the background knowledge predicates zincsite/2, hacc/2, and dist/5. (**b**) The inductive logic programming system Progol automatically translates (**a**) into the "Sternberg English" rule, which can be easily read by human experts. (**c**) *A* molecule with the active site indicated by the atoms *B*, *C*, *D*, and *E* (Image courtesy of Finn et al. 1998)

Learning from Graphs

A graph is a mathematical structure consisting of a set of nodes V and a set of edges $E \subseteq V^2$ between those nodes. The set of edges is by definition a binary relation defined over the nodes. Hence, for any learning problem where the relationships between examples can be described using a single binary relation, the training set can be represented straightforwardly as a graph. This setting covers a wide range of relational learning tasks, for example, web mining (the set of links between pages is a binary relation), social network analysis, etc. Non-binary relationships can be represented as hypergraphs; in a hypergraph, edges are defined as subsets of V of arbitrary size, rather than elements of V^2.

In graph-based learning systems, there is a clear distinction between approaches that learn from examples with external relationships, where the whole data set is represented as a single graph and each node is an example, and individual-centered approaches, where each example by itself is a graph. In the first kind of approaches, the goal is often to predict properties of existing nodes or edges, to predict the existence or non-existence of edges ("link discovery"), to predict whether two nodes actually refer to the same object ("node identification"), detection of subgraphs that frequently occur in the graph, etc. When learning from multiple graphs, a typical goal is to learn a model for classifying the graphs, to find frequent substructures (where frequency is defined as the number of graphs a subgraphs occurs in), etc.

Compared to other methods for relational learning, graph-based methods typically focus more on the structure of the graph, and less on properties of single nodes. They may take node and edge labels into account, but typically do not allow for more elaborate information to be associated with each node.

▶ Graph mining methods are often more efficient than other relational mining methods because they avoid certain kinds of overhead, but are typically still NP-complete, as they generally rely on subgraph isomorphism testing. Nevertheless, researchers have been able to significantly improve efficiency or even avoid NP-completeness by looking only for linear or tree-shaped patterns, or by restricting the graphs analyzed to a relatively broad subclass. As an example, Horváth et al. (2006) show that a large majority of molecules belong to the class of outerplanar graphs, and propose an efficient algorithm for subgraph isomorphism testing in this class.

More information about mining graph data can be found in the ▸ graph mining entry in this encyclopedia, or in Cook and Holder (2007) and Washio and Motoda (2003).

Multi-relational Data Mining

Multi-relational data mining approaches relational learning from the relational database point of view. The term "multi-relational" refers to the fact that from the database perspective, one learns from information spread over multiple tables or relations, as opposed to ▸ attribute-value learning, where one learns from a single table.

Multi-relational data mining systems tightly integrate with relational databases. Mainly rule and decision tree learners have been developed in this setting. Because practical relational databases may be huge, most of these systems pay much attention to efficiency and scalability, and use techniques such as sampling and precomputation (e.g., materializing views). An example of a scalable and efficient multi-relational rule learning system is CrossMine (Yin et al. 2006).

An alternative approach to relational learning and multi-relational data mining is ▸ propositionalization. Propositionalization consists of automatically converting the relational representation into an attribute-value representation and then using attribute-value data mining algorithms on the resulting representation. An important line of research within multi-relational data mining investigates how database approaches can be used to this end. Database oriented propositionalization creates a view in which each example is represented by precisely one row. Information from related entities is incorporated into this row by adding derived attributes, computed by means of aggregation. In the movie database (Example 2), the view representing movies could include aggregated attributes such as the number of actors starring in the movie. A comparison of propositionalization approaches is presented by Krogel et al. (2003), and a discussion of them is also included in this volume.

Finally, note that most inductive logic programming systems are directly applicable to multi-relational data mining by representing each relational table as a predicate. This is possible because the relational representation is essentially a subset of first-order logic (known as datalog). Much research on multi-relational data mining was developed within the ILP community (Džeroski and Lavraè 2001).

Statistical Relational Learning/Probabilistic Logic Learning

Research on relational learning, especially in the beginning, has largely focused on how to handle the relational structure of the data, and ignored aspects such as uncertainty. Indeed, the databases handled in multi-relational data mining, or the knowledge assumed given in inductive logic programming, are typically assumed to be deterministic. With the rise of probabilistic representations and algorithms within machine learning has come an increased interest in enabling relational learners to cope with uncertainty in the input data. This goal has been approached from at least two different directions: statistical learning approaches have been extended toward the relational setting, giving rise to the area of ▸ statistical relational learning, whereas inductive logic programming researchers have investigated how to extend their knowledge representation and learning algorithms to cater for probabilistic information, referring to this research area as probabilistic logic learning. While there are some differences in terminology and approaches, both research areas essentially address the same research question, namely how to integrate relational and probabilistic learning.

Among the best known approaches for statistical relational learning is the learning of probabilistic relational models (PRMs, Džeroski and Lavraè 2001, Chap. 13). PRMs extend Bayesian networks to the relational representation used in relational databases. PRMs model the joint probability distribution over the non-key attributes in a relational database schema. Similar to Bayesian networks, PRMs are ▶ graphical models. Each attribute corresponds to a node and direct dependencies are modeled by directed edges. Such edges can connect attributes from different entity types that are (indirectly) related (such a relationship is called a "slot chain"). Inference in PRMs occurs by constructing a ▶ Bayesian network by instantiating the PRM with the data in the database and performing the inference in the latter. To handle 1:N relationships in the Bayesian network, PRMs make use of aggregation, similar to the propositionalization techniques mentioned above.

Bayesian logic programs (BLPs) (Kersting 2006) aim at combining the inference power of Bayesian networks with that of first-order logic reasoning. Similar to PRMs, the semantics of a BLP is defined by translating it to a Bayesian network. Using this network, the probability of a given interpretation or the probability that a given query yields a particular answer can be computed.

The acyclicity requirement of Bayesian networks carries over to representations such as PRMs and BLPs. Markov logic networks (MLNs) (Richardson and Domingos 2006) upgrade ▶ Markov networks to first-order logic and allow networks with cycles. MLNs are defined as sets of weighted first-order logic formulas. These are viewed as "soft" constraints on logical interpretations: the fewer formulas a given interpretation violates, the higher its probability. The weight determines the contribution of a given formula: the higher its weight, the greater the difference in log probability between an interpretation that satisfies the formula and one that does not, other things being equal. The Alchemy system implements structure and parameter learning for MLNs.

More specific statistical learning techniques such as Naïve Bayes and Hidden Markov Models have also been upgraded to the relational setting. More information about such algorithms and about statistical relational learning in general can be found in Getoor and Taskar (2007) and Kersting (2006).

In probabilistic logic learning, two types of semantics are distinguished (De Raedt and Kersting 2003): the model theoretic semantics and the proof theoretic semantics. Approaches that are based on the model theoretic semantics define a probability distribution over interpretations and extend probabilistic attribute-value techniques, such as Bayesian networks and Markov networks, while proof theoretic semantics approaches define a probability distribution over proofs and upgrade, e.g., stochastic context free grammars.

Example 5 Consider the case where each example is a sentence in natural language. In this example, a model theoretic approach would define a probability distribution directly over sentences. A proof theoretic approach would define a probability distribution over "proofs," in this case possible parse trees of the sentence (each sentence may have several possible parses). Note that the proof theoretic view is more general in the sense that the distribution over sentences can be computed from the distribution over proofs.

Stochastic logic programs (SLPs) (Muggleton 1996) follow most closely the proof theoretic view and upgrade stochastic context free grammars to first-order logic. SLPs are logic programs with probabilities attached to the clauses such that the probabilities of clauses with the same head sum to 1.0. The probability of a proof is then computed as the product of the probabilities of the clauses that are used in the proof. PRISM (Sato and Kameya 1997) follows a related approach where the probabilities are defined on ground facts.

Like with standard graphical models, learning algorithms may include both parameter learning (estimating the probabilities) and structure learning (learning the program). For most frameworks mentioned above, such techniques have been or are being developed.

For a more detailed treatment of statistical relational learning and probabilistic logic learning, we refer to the entry on statistical relational learning in this volume, and to several reference works (De Raedt and Kersting 2003; Getoor and Taskar 2007; Kersting 2006; De Raedt et al. 2008).

Relational Reinforcement Learning

Relational reinforcement learning (RRL) (Džeroski et al. 2001; Tadepalli et al. 2004) is reinforcement learning upgraded to the relational setting. Reinforcement learning is concerned with how an agent should act in a given environment to maximize its accumulated reward. In RRL, both the state of the environment and the actions are represented using a relational representation, typically in the form of a logic program.

Much research in RRL focuses on Q-learning, which represents the knowledge of the agent by means of a Q-function mapping state–action pairs to real values. During exploration, the agent selects in each state the action that is ranked highest by the Q-function. The Q-function is typically represented using a relational regression technique. Several techniques, such as relational regression trees, relational instance based learning, and relational kernel based regression have been considered in this context. Note that the regression algorithms must be able to learn incrementally: each time the agent receives a new reward, the Q-function must be incrementally updated for the episode (sequence of state-action pairs) that led to the reward. Due to the use of relational regression techniques, the agent is able to generalize over states: it will perform similar actions in similar states and therefore scales better to large application domains.

More recent topics in RRL include how expert knowledge can be provided to the agent in the form of guidance, and how learned knowledge can be transferred to related domains ("transfer learning"). More details on these techniques and more specific information on the topic of relational reinforcement learning can be found in its corresponding encyclopedia entry and in the related entry on ▶ symbolic dynamic programming, as well as in references Džeroski et al. (2001) and Tadepalli et al. (2004).

Cross-References

▶ Inductive Logic Programming
▶ Multi-relational Data Mining
▶ Relational Reinforcement Learning

Recommended Reading

Most of the topics covered in this entry have more detailed entries in this encyclopedia, namely "Inductive Logic Programming," "Graph Mining," "Relational Data Mining," and "Relational Reinforcement Learning." These entries provide a brief introduction to these more specific topics and appropriate references for further reading. Direct relevant references to the literature include the following. A comprehensive introduction to ILP can be found in De Raedt's book (De Raedt 2008) on logical and relational learning, or in the collection edited by Džeroski and Lavraè (2001) on relational data mining. Learning from graphs is covered by Cook and Holder (2007). Džeroski and Lavraè (2001) is also a good starting point for reading about multi-relational data mining, together with research papers on multi-relational data mining systems, for instance, Yin et al. (2006), who present a detailed description of the CrossMine system. Statistical relational learning in general is covered in the collection edited by Getoor and Taskar (2007), while De Raedt and Kersting (2003) and De Raedt et al. (2008) present overviews of approaches originating in logic-based learning. An overview of relational reinforcement learning can be found in Tadepalli et al. (2004).

Bratko I (2000) Prolog programming for artificial intelligence, 3rd edn. Addison-Wesley, Reading

Cook DJ, Holder LB (2007) Mining graph data. Wiley, Hoboken

De Raedt L (2008) Logical and relational learning. Springer, Berlin

De Raedt L, Kersting K (2003) Probabilistic logic learning. SIGKDD Explor 5(1):31–48

De Raedt L, Frasconi P, Kersting K, Muggleton S (2008) Probabilistic inductive logic programming. Springer, Berlin

Džeroski S, De Raedt L, Driessens K (2001) Relational reinforcement learning. Mach Learn 43:7–52

Džeroski S, Lavraè N (eds) (2001) Relational data mining. Springer, Berlin

Finn P, Muggleton S, Page D, Srinivasan A (1998) Pharmacophore discovery using the inductive logic programming system PROGOL. Mach Learn 30:241–270

Getoor L, Taskar B (2007) Introduction to statistical relational learning. MIT Press, Cambridge

Horváth T, Ramon J, Wrobel S (2006) Frequent subgraph mining in outerplanar graphs. In: Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, pp 197–206

Jensen D, Neville J (2002) Linkage and autocorrelation cause feature selection bias in relational learning. In: Proceeding of the 19th international conference on machine learning, University of New South Wales, Sydney. Morgan Kaufmann, San Francisco, pp 259–266

Jensen D, Neville J, Gallagher B (2004) Why collective inference improves relational classification. In: Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining, Philadelphia. ACM, New York, pp 593–598

Kersting K (2006) An inductive logic programming approach to statistical relational learning. IOS Press, Amsterdam

Krogel M-A, Rawles S, Železný F, Flach P, Lavraè N, Wrobel S (2003) Comparative evaluation of approaches to propositionalization. In: Proceedings of the 13th international conference on inductive logic programming, Szeged. Springer, Berlin, pp 194–217

Lloyd JW (2003) Logic for learning. Springer, Berlin

Muggleton S (1996) Stochastic logic programs. In: De Raedt L (ed) Advances in inductive logic programming. IOS Press, Amsterdam, pp 254–264

Richardson M, Domingos P (2006) Markov logic networks. Mach Learn 62(1–2):107–136

Sato T, Kameya Y (1997) PRISM: a symbolic-statistical modeling language. In: Proceedings of the 15th international joint conference on artificial intelligence (IJCAI 97), Nagoya. Morgan Kaufmann, San Francisco, pp 1330–1335

Tadepalli P, Givan R, Driessens K (2004) Relational reinforcement learning: an overview. In: Proceeding of the ICML'04 workshop on relational reinforcement learning, Banff, pp 1–9

Washio T, Motoda H (2003) State of the art of graph-based data mining. SIGKDD Explor 5(1):59–68

Yin X, Han J, Yang J, Yu PS (2006) Efficient classification across multiple database relations: a CrossMine approach. IEEE Trans Knowl Data Eng 18(6): 770–783

Relational Regression Tree

▶ First-Order Regression Tree

Relational Reinforcement Learning

Kurt Driessens
Maastricht University, Maastricht, The Netherlands

Synonyms

Learning in worlds with objects; Reinforcement learning in structured domains

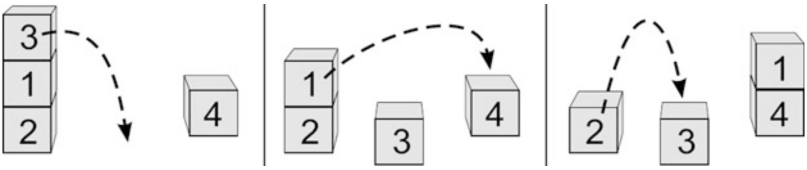
Definition

Relational reinforcement learning is concerned with learning behavior or control policies based on a numerical feedback signal, much like standard reinforcement learning, in complex domains where states (and actions) are largely characterized by the presence of objects, their properties, and the existing relations between those objects. Relational reinforcement learning uses approaches similar to those used for standard reinforcement learning, but extends these with methods that can abstract over specific object identities and exploit the structural information available in the environment.

Motivation and Background

▶ Reinforcement learning is a very attractive machine learning framework, as it tackles, in a sense, the whole artificial intelligence problem at a small scale: an agent acts in an unknown environment and has to learn how to behave optimally by reinforcement, i.e., through rewards and punishment. Reinforcement learning has produced some impressive and promising results. However, the applicability of reinforcement learning has been greatly limited by its difficulty in dealing with large problem spaces and its inability to generalize the learned knowledge to new but related problem domains.

Relational Reinforcement Learning, Fig. 1 Structure of the RRL system



While standard reinforcement learning methods represent the learning environment as a set of unrelated states or, when using ▸ attribute-value representations, as a vector space consisting of a fixed number of independent dimensions, humans tend to think about their environment in terms of objects, their properties, and the relations between them. Examples of objects in everyday life are chairs, people, streets, trees, etc. This representation allows people to treat or use most of the new objects that they encounter correctly, without requiring training time to learn (again) how to use them. For example, people are able to drink their coffee from any cup that will hold it, even if they have never encountered that specific cup before, because they already have experience with drinking their coffee from other cup-type objects. Standard reinforcement learning agents do not have this ability. Their state and action representations do not allow them to abstract away from specific object identities and recognize them as a type of object they are already accustomed to.

Relational reinforcement learning tries to overcome this problem by representing states of the learning agent's environment as sets of objects, their properties, and the relationships between them, similar to the approaches used in ▸ relational learning and ▸ inductive logic programming. These structural representations make it possible for the relational reinforcement learning agent to abstract away from specific identities of objects and often also from the amount of objects present, the exact learning environment, or even the specific task to be performed.

The term "relational reinforcement learning" was introduced by Džeroski et al. (1998) when they first teamed the Q-learning algorithm with a first-order regression algorithm. From then on, relational reinforcement learning gained a large amount of interest.

Structure of the Learning System

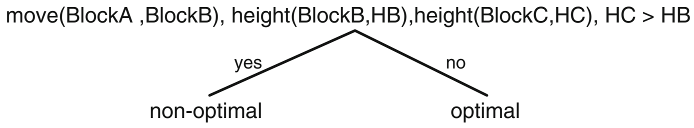
In principle, the structure of a relational reinforcement learning system is very similar to that of standard reinforcement learning systems (Fig. 1). At a high level, the learning agent interacts with an environment by performing actions that influence that environment, and the environment provides the learning agent with a description of its current state and a numerical feedback of the performance of the agent. The goal of the agent is to maximize some cumulative form of this feedback signal. The major difference between standard reinforcement learning and relational reinforcement learning is the representation of the state–action–space. Relational reinforcement learning works on ▸ Markov decision processes where states and actions have been relationally factored, so-called relational Markov decision processes (RMDPs).

An RMDP can be defined as follows:

Definition 1 (Relational Markov Decision Process) Let P_S be a set of state-related predicates, P_A a set of action-related predicates, and C a set of constants in a logic Λ. Let $\mathcal{B}$ be a theory defined in that logic.

An RMDP is defined as $< S, A, T, R >$, where $S \equiv \{s \subset H^{P_S \cup C} | s \models \mathcal{B}\}$ represents the set of states; $A \equiv \{a \subset H^{P_A \cup C} | a \models \mathcal{B}\}$ represents the set of actions, in which H^X is the set of facts that can be constructed given the symbols in X; and T and R represent the transition probabilities and reward function, respectively: $T : S \times A \times S \to [0, 1]$ and $R : S \to \mathbb{R}$.

In less formal language, this means that the states and actions in an RMDP are represented using a set of constants C and a set of predicates P_S and P_A, respectively, and constrained by a background theory $\mathcal{B}$. This means that the background theory $\mathcal{B}$ defines which states are possible


move(BlockA ,BlockB), height(BlockB,HB),height(BlockC,HC), HC > HB
yes
no
non-optimal
optimal

Relational Reinforcement Learning, Fig. 2 Example state–action pairs in the blocks world

in the domain and which actions can be executed in which states.

The following example illustrates these concepts. Consider the blocks world depicted in Fig. 2. To represent this environment in first-order logic, one could use:

- State-related predicates: $P_S = \{\text{on/2, clear/1}\}$
- Action-related predicate: $P_A = \{\text{move/2}\}$
- Constants: $C = \{1,2,3,4,\text{floor}\}$

The set of facts $H^{P_S \cup C}$ would then include, for example, $on(1,2)$, $on(4, floor)$, and $clear(2)$ but also $on(3,3)$ and $on(floor, 2)$. To constrain the possible states to those that actually make sense in a standard, i.e., real-world view of the blocks world, the theory $\mathcal{B}$ can include rules to make states that include these kinds of facts impossible. For example, to make sure that a block cannot be on top of itself, $\mathcal{B}$ could include the following constraint:

$$\texttt{false} \leftarrow on(X, X).$$

One can also include more extensive rules to define the exact physics of the blocks world that one is interested in. For example, by including

$$\texttt{false} \leftarrow on(Y, X), on(Z, X), X \neq floor,$$
$$Y \neq Z$$

as part of the theory $\mathcal{B}$, one can exclude states where two blocks are on top of the same block. The action space given by $H^{P_A \cup C}$ consists of facts such as $move(3,2)$ and $move(floor, 1)$ and can be constrained by rules such as

$$\texttt{false} \leftarrow move(floor, X),$$

which makes sure that the floor cannot be placed on top of a block.

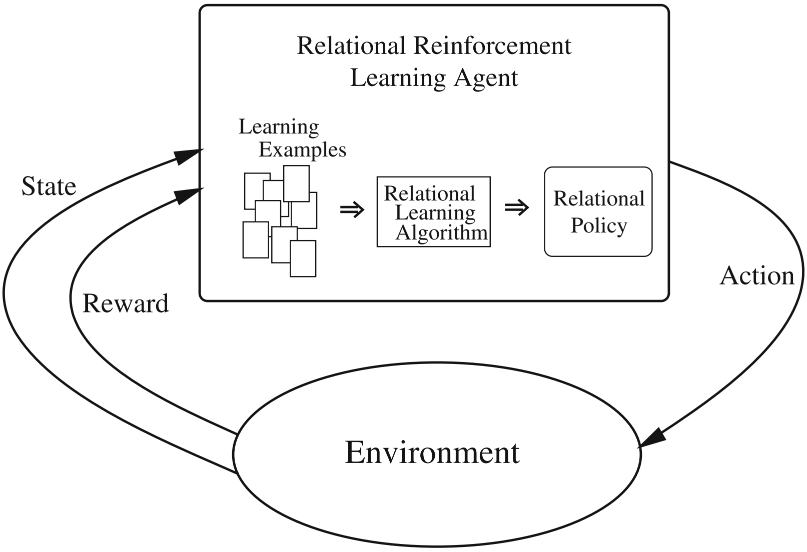
The leftmost state–action pair of Fig. 2 can be fully specified by the following set of facts (state description on the left, action on the right):

on(2,floor).	clear(3).
on(1,2).	clear(4).
on(3,1).	clear(floor).
on(4,floor).	move(3,floor).

One can easily generalize over specific states and create abstract states (or state–action pairs) that represent sets of states (or state–action pairs) by using variables instead of constants and by listing only those parts of states and actions that hold for each element of the abstract state (or state–action pair). For example, the abstract state "$on(1,2), on(2, floor)$" represents all states in which block 1 is on top of block 2, which in turn is on the floor. The abstract state does not specify the locations of any other blocks. Of the three states depicted in Fig. 2, the set of states represented by the abstract state would include the left and middle states. Abstract states can also be represented by using variables when one does not want to specify the location of any specific block, but wants to focus on structural aspects of the states and actions. The abstract state–action pair "$move(X, Y), on(Y, floor)$" represents all state–action pairs where a block is moved on top of another block that is on the floor, for example, the middle and right state–action pairs of Fig. 2.

Benefits of Relational Reinforcement Learning

We already stated that the real world is made of interacting objects or at least that humans often think about the real world as such. Relational reinforcement learning allows this same representation to be used by reinforcement learning agents, which in turn leads to more human-interpretable learning results.



Relational Reinforcement Learning, Fig. 3 Simple relational policy for stacking any number of blocks

As a consequence of the used logical or relational representation of states and actions, the results learned by a relational reinforcement learning agent can be reused more easily when some of the parameters of the learning task change. Because relational reinforcement learning algorithms try to solve the problem at hand at an abstract level, the solutions will often carry over to different instantiations of that abstract problem. For example, the resulting policies learned by the RRL system (Driessens 2004) discussed below, a very simple example of which is shown in Fig. 3, often generalize over domains with a varying number of objects. If only actions which lead to the *optimal* leaf are executed, the shown policy tree will organize any number of blocks into a single stack.

As another example of this, the relational approximate policy iteration approach, also discussed below, is able to learn task-specific control knowledge from random walks in the environment. By treating the resulting state of such a random walk as a goal state and generalizing over the specifics of that goal (and the rest of the random walk), relational approximate policy iteration can learn domain-specific, but goal-independent, policies. This generalization of the policy is accomplished by parameterization of the goal and focusing on the relations between objects in the goal, states, and actions when representing the learned policy.

Another practical benefit of relational reinforcement learning lies in the field of inductive transfer. Transfer learning is concerned with the added benefits of having experience with a related task when being confronted with a new one. Because of the structural representation of learned results, the transfer of knowledge learned by relational reinforcement learning agents can be accomplished by recycling those parts of the results that still hold valid information for the new task. Depending on the relation between the two tasks, this can yield substantial benefits concerning the required training experience.

The use of first-order logic as a representational language in relational reinforcement learning also allows the integration of reasoning methods with traditional reinforcement learning approaches. One example of this is ▸ symbolic dynamic programming, which uses logical regression to compute necessary preconditions that allow an agent to reach certain goals. This same integration allows the use of search or planning knowledge as background information to extend the normal description of states and actions.

Example Relational Reinforcement Learning Approaches

Relational Q-Learning

Relational reinforcement learning was introduced with the development of the RRL system (Džeroski et al. 1998). This is a Q-learning system that employs a relational regression algorithm to generalize the Q-table used by standard Q-learning algorithms into a Q-function. The differences with a standard Q-learning agent are mostly located inside the learning agent. One important difference is the agent's representation of the current state. In relational reinforcement learning, this representation contains structural or relational information about the environment.

Inside the learning agent, the information consisting of encountered states, chosen actions, and the associated rewards is translated into learning examples. These examples are then processed by a relational learning system that produces a relational Q-function and/or policy as a result. The relational representation of the Q-function allows the RRL system to use the structural properties of states and actions when assigning a Q-value to them.

Several relational regression approaches have been developed and applied in this context. While the original approach used an off-the-shelf relational regression algorithm that processed the learning examples in batch and had to be restarted to be able to process newly available learning experiences, a number of incremental algorithms have been developed for use in relational reinforcement learning since then. These include an incremental first-order regression tree algorithm, incremental relational instance-based regression, kernel-based regression that uses Gaussian processes, and graph kernels and algorithms that include combinations of the above (Driessens 2004).

It is possible to translate the learned Q-function approximations into a function that directly represents its policy. Using the values predicted by the learned Q-function, one can generate learning examples that represent state–action pairs and label them as either part of the learned policy or not. This results in a binary classification problem that can be handled by a supervised relational learning algorithm such as TILDE (Blockeel and De Raedt 1998), as used to produce first-order decision tree policies in the original work. This technique is known as P-learning. It exhibits better generalization performance across related learning problems than the Q-learning approach described above. Other than the aforementioned first-order decision trees, rule-based learners have also been applied to this kind of policy learning.

Nonparametric Policy Gradients

Nonparametric policy gradients (Kersting and Driessens 2008), also a model-free approach, apply Friedmann's gradient boosting (Friedman 2001) in an otherwise standard policy gradient approach for reinforcement learning. To avoid having to represent policies using a fixed number of parameters, policies are represented as a weighted sum of regression models grown in a stage-wise optimization. (This allows the number of parameters to grow as the experience of the learner increases, hence the name nonparametric.) While this does not make nonparametric policy gradients a technique specifically designed for relational reinforcement learning, it allows, like the relational Q-learning approach described above, the use of relational regression models and is not constrained to the attribute-value setting of standard policy gradients.

The idea behind the approach is that instead of finding a single, highly accurate policy, it is easier to find many rough rules of thumb of how to change the way the agent currently acts. The learned policy is represented as

$$\pi(s,a) = \frac{e^{\Psi(s,a)}}{\sum_b e^{\Psi(s,b)}} ,$$

where instead of assuming a linear parameterization for Ψ as is done in standard policy gradients, it is assumed that Ψ will be represented by a linear combination of functions. Specifically, one starts with some initial function Ψ_0, e.g., based on the zero potential, and iteratively adds corrections $\Psi_m = \Psi_0 + \Delta_1 + \cdots + \Delta_m$. In contrast to the standard gradient approach, Δ_m here denotes the so-called functional gradient, which is sampled during interaction with the environment and then generalized by an off-the-shelf regression algorithm.

The advantages of policy gradients over value-function techniques are that they can learn non-deterministic policies and that convergence of the learning process can be guaranteed, even when using function approximation (Sutton et al. 2000). Experimental results show that nonparametric policy gradients have the potential to significantly outperform relational Q-learning (Kersting and Driessens 2008).

Relational Approximate Policy Iteration

A different approach, which also directly learns a policy, is taken in relational approximate policy iteration (Fern et al. 2006). Like standard policy iteration (Sutton and Barto 1998), the approach iteratively improves its policy through interleaving evaluation and improvement steps. In contrast to standard policy iteration, it uses a policy language bias and a generalizing policy function.

Instead of building a value-function approximation for each policy evaluation step, relational approximate policy iteration evaluates the current policy and its closely related neighbors by sampling the state–action–space through a technique called policy rollout. This technique generates a set of trajectories from a given state, by executing every possible action in that state and following the current policy for a number of steps afterward. (It is also possible to improve convergence speed by following the next policy.) These trajectories and their associated costs result in a number of learning examples – one for each possible action in each selected state – that can be used, together with the policy language bias to generate the next, improved policy.

Because every possible action in each sampled state needs to be evaluated, this approach does require a model or a resettable simulator of the environment. However, relational approximate policy iteration has been shown to work well for learning domain-specific control knowledge and performs very well on planning competition problems.

Relational Cross Entropy Policy Search

The most recent addition to direct relational policy search uses the cross entropy method to evaluate, select, expand, and combine those pieces of a modular policy that lead to high-performance behavior (Sarjant et al. 2014). The policy pieces are singular condition-action rules. The rules are constructed and adapted automatically using a partial model of the environment inferred from interactions with that environment. The model defines the minimal conditions needed to take an action, the possible specialization conditions per rule, and a set of simplification rules to remove redundant and illegal rule conditions. Rule construction and specialization follow a principled approach toward exploration of the rule space by beginning with the relative least general generalization (RLGG) rules and then exploring incremental specializations of interesting rules.

The cross entropy method (CEM) (Rubinstein 1997) is used to find these interesting rules. CEM tunes the selection probabilities of the rules according to the performance of policies that they participate in. Rules with high selection probabilities also become candidates for specialization, possibly giving rise to even better rules.

The resulting systems can learn behavior that is competitive to specialized approaches on complex tasks, while the built-in simplification of the rules and CEM bias toward compact policies result in comprehensive and effective relational policies.

Symbolic Dynamic Programming

In contrast to the previous techniques, ▶ symbolic dynamic programming (SDP) does not learn a policy through exploration of the environment. Instead, it is a model-based approach that uses knowledge about preconditions and consequences of actions to compute the fastest way to reach a given goal. Like other dynamic programming techniques, SDP starts from the goal the agent wants to reach and reasons backward to find the policy that is needed to reach that goal. In contrast to other dynamic programming techniques, it does not solve specific instantiations of the problem domain, but instead solves the problem at an abstract level, thereby solving it for all possible instantiations of the problem at once.

SDP treats the required goal conditions as an abstract state definition. Because pre- and post-conditions of actions are known, SDP can compute the necessary conditions that allow actions to reach the abstract goal state. These conditions define abstract states from which it is possible to reach a goal state in one step. Starting from these abstract states, the same approach can be used to discover abstract states that allow the goal to be reached in two steps and so on.

This approach was first proposed by Boutilier et al. (2001), implemented as a working system by Kersting et al. (2004), and later improved upon by Sanner and Boutilier (2005). This last approach won second place in the probabilistic programming competition at ICAPS in 2006.

Cross-References

- ▸ Hierarchical Reinforcement Learning
- ▸ Inductive Logic Programming
- ▸ Model-Based Reinforcement Learning
- ▸ Policy Search
- ▸ Q-Learning
- ▸ Reinforcement Learning
- ▸ Relational Learning
- ▸ Symbolic Dynamic Programming
- ▸ Temporal Difference Learning

Further Information

The field of relational reinforcement learning has given rise to a number of PhD dissertations in the last few years (Croonenborghs 2009; Driessens 2004; van Otterlo 2008; Sanner 2008). The dissertation of Martijn van Otterlo resulted in a book (van Otterlo 2009) which provides a recent and reasonably complete overview of the relational reinforcement learning research field. Other publications that present an overview of relational reinforcement learning research include the proceedings of the two workshops on representational issues in (relational) reinforcement learning at the International Conferences on Machine Learning in 2004 and 2005 (Driessens et al. 2005; Tadepalli et al. 2004).

Recommended Reading

Blockeel H, De Raedt L (1998) Top-down induction of first order logical decision trees. Artif Intell 101(1–2):285–297

Boutilier C, Reiter R, Price B (2001) Symbolic dynamic programming for first-order MDPs. In: Proceedings of the 17th international joint conference on artificial intelligence (IJCAI-2001), Seattle, pp 690–700

Croonenborghs T (2009) Model-assisted approaches for relational reinforcement learning. Ph.D. thesis, Department of Compute Science, Katholieke Universiteit Leuven

Driessens K (2004) Relational reinforcement learning. Ph.D. thesis, Department of Computer Science, Katholieke Universiteit Leuven

Driessens K, Fern A, van Otterlo M (eds) (2005) Proceedings of ICML-2005 workshop on rich representation for reinforcement learning, Bonn

Džeroski S, De Raedt L, Blockeel H (1998) Relational reinforcement learning. In: Proceedings of the 15th international conference on machine learning (ICML-1998), San Francisco. Morgan Kaufmann, Madison, pp 136–143

Džeroski S, De Raedt L, Driessens K (2001) Relational reinforcement learning. Mach Learn 43:7–52

Fern A, Yoon S, Givan R (2006) Approximate policy iteration with a policy language bias: solving relational Markov decision processes. J Artif Intell Res 25:85–118

Friedman J (2001) Greedy function approximation: a gradient boosting machine. Ann Stat 29:1189–1232

Kersting K, Driessens K (2008) Non-parametric policy gradients: a unified treatment of propositional and relational domains. In: McAllum A, Roweis S (eds) Proceedings of the 25th international conference on machine learning (ICML 2008), Helsinki, pp 456–463

Kersting K, van Otterlo M, De Raedt L (2004) Bellman goes relational. In: Proceedings of the twenty-first international conference on machine learning (ICML-2004), Banff, pp 465–472

Rubinstein RY (1997) Optimization of computer simulation models with rare events. Eur J Oper Res 99(1):89–112

Sanner S (2008) First-order decision-theoretic planning in structured relational environments. Ph.D. thesis, Department of Compute Science, University of Toronto

Sanner S, Boutilier C (2005) Approximate linear programming for first-order MDPs. In: Proceedings of the 21st conference on Uncertainty in AI (UAI), Edinburgh

Sarjant S (2013) Policy search based relational reinforcement learning using the cross-entropy method. Ph.D. thesis, Department of Computer Science, University of Waikato

Sarjant S, Pfahringer B, Driessens K, Smith T (2014) A Direct Policy-Search Algorithm for Relational Reinforcement Learning. In: Proceedings of the 25th international conference on inductive logic programming (ILP 2013), Rio de Janeiro, pp 76–92

Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT, Cambridge

Sutton RS, McAllester D, Singh S, Mansour Y (2000) Policy gradient methods for reinforcement learning with function approximation. In: Advances in neural information processing systems, vol 12. MIT, Cambridge, pp 1057–1063

Tadepalli P, Givan R, Driessens K (eds) (2004) Proceedings of the ICML-2004 workshop on relational reinforcement learning, Banff

van Otterlo M (2008) The logic of adaptive learning. Ph.D. thesis, Centre for Telematics and Information Technology, University of Twente

van Otterlo M (2009) The logic of adaptive behavior: knowledge representation and algorithms for adaptive sequential decision making under uncertainty in first-order and relational domains. IOS Press, Amsterdam

Relational Value Iteration

► Symbolic Dynamic Programming

Relationship Extraction

► Link Prediction

Relevance Feedback

Relevance feedback provides a measure of the extent to which the results of a search match the expectations of the user who initiated the query. Explicit feedback require users to assess relevance by choosing one out of a number of choices, or to rank documents to reflect their perceived degree of relevance. Implicit feedback is obtained by monitoring user's behavior such as time spent browsing a document, amount of scrolling performed while browsing a document, number of times a document is visited, etc. Relevance feedback is one the techniques used to support query reformulation and turn the search into an iterative and interactive process.

Cross-References

► Search Engines: Applications of ML

Representation Language

► Hypothesis Language

Reservoir Computing

Risto Miikkulainen
Department of Computer Science, The University of Texas at Austin, Austin, TX, USA

Synonyms

Echo state network; Liquid state machine

Definition

Reservoir computing is an approach to sequential processing where recurrency is separated from the output mapping (Jaeger 2003; Maass et al. 2002). The input sequence activates neurons in a recurrent neural network (a reservoir, where activity propagates as in a liquid). The recurrent network is large, nonlinear, randomly connected, and fixed. A linear output network receives activation from the recurrent network and generates the output of the entire machine. The idea is that if the recurrent network is large and complex enough, the desired outputs can likely be learned as linear transformations of its activation. Moreover, because the output transformation is linear, it is fast to train. Reservoir computing has been successful in particular in speech and language processing and vision and cognitive neuroscience.

Recommended Reading

Jaeger H (2003) Adaptive nonlinear system identification with echo state networks. In: Becker S, Thrun S, Obermayer K (eds) Advances in neural information processing systems, vol 15. MIT, Cambridge, pp 593–600

Maass W, Natschlaeger T, Markram H (2002) Real-time computing without stable states: a new framework for neural computation based on perturbations. Neural Comput 14:2531–2560

Resubstitution Estimate

Resubstitution estimates are estimates that are derived by applying a model to the ▶ training data from which it was learned. For example, *resubstitution error* is the error of a model on the training data.

Cross-References

▶ Model Evaluation

Reward

In most *Markov decision process* applications, the decision-maker receives a *reward* each period. This reward can depend on the current *state*, the *action* taken, and the next state and is denoted by $r_t(s, a, s')$.

Reward Selection

▶ Reward Shaping

Reward Shaping

Eric Wiewiora
University of California, Sydney, NSW,
Australia

Synonyms

Heuristic rewards; Reward selection

Definition

Reward shaping is a technique inspired by animal training where supplemental rewards are provided to make a problem easier to learn. There is usually an obvious natural reward for any problem. For games, this is usually a win or loss. For financial problems, the reward is usually profit. Reward shaping augments the natural reward signal by adding additional rewards for making progress toward a good solution.

Motivation and Background

Reward shaping is a method for engineering a reward function in order to provide more frequent feedback on appropriate behaviors. It is most often discussed in the ▶ reinforcement learning framework. Providing feedback is crucial during early learning so that promising behaviors are tried early. This is necessary in large domains, where reinforcement signals may be few and far between.

A good example of such a problem is chess. The objective of chess is to win a match, and an appropriate reinforcement signal should be based on this. If an agent were to learn chess without prior knowledge, it would have to search for a great deal of time before stumbling onto a winning strategy. We can speed up this process by rewarding the agent more frequently. One possibility is to reward the learner for capturing enemy pieces, and punish the learner for losing pieces. This new reward creates a much richer learning environment, but also runs the risk of distracting the agent from the true goal (winning the game).

Another domain where feedback is extremely important is in robotics and other real-world applications. In the real world, learning requires a large amount of interaction time, and may be quite expensive. Mataric noted that in order to mitigate “thrashing” (repeatedly trying ineffective actions) rewards should be supplied as often as possible (Mataric 1994).

If a problem is inherently described by sparse rewards, it may be difficult to change the reward structure without disrupting progress to the original goal. The behavior that is optimal with a richer reward function may be quite different from the intended behavior, even if relatively small shaping rewards are added. A classic example of this is found in Randlov and Alstrom (1998). While training an agent to control a bicycle simulation, they rewarded an agent whenever it moved toward a target destination. In response to this reward, the agent learned to ride in a tight circle, receiving reward whenever it moved in the direction of the goal.

Theory

We assume a reinforcement learning framework. For every time step t, the learner observes state s_t, takes action a_t, and receives reward r_t. The goal of reinforcement learning is to find a policy $\pi(s)$ that produces actions that optimize some long-term measurement of reward. We define the value function for every state as the expected infinite horizon discounted reward

$$V(s) = \max_{\pi} \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_t = \pi(s_t)\right],$$

where γ is the discount rate. A reinforcement learner's goal is to learn a good estimate of $V(s)$, and to use this estimate to choose a good policy.

A natural reward source should be fairly obvious from the problem at hand. Financial problems should use net monetary gain or loss as reward. Games and goal-directed problems should reward winning the game or reaching the goal. It is usually advantageous to augment this natural reward with a shaping reward f_t. We define the augmented value function V' for the reinforcement learning problem with shaping rewards

$$V'(s) = \max_{\pi'} \mathrm{E}\Bigg[\sum_{t=0}^{\infty} \gamma^t (r_t + f_t)|s_0 \\ = s, a_t = \pi'(s_t)\Bigg].$$

Ideally, the policy that optimizes the augmented value function will differ much from the previous optimal policy.

Constructing an appropriate shaping reward system is inherently a problem-dependent task, though a line of research aids in the implementation of these reward signals. *Potential-based shaping* provides a formal framework for translating imperfect knowledge of the relative value of states and actions into a shaping reward.

Potential-Based Shaping

Ng et al. proposed a method for adding shaping rewards in a way that guarantees the optimal policy maintains its optimality (Ng et al. 1999). They define a potential function $\Phi()$ over the states. The shaping reward f for transitioning from state s to s' is defined as the discounted change in this state potential:

$$f(s, s') = \gamma\Phi(s') - \Phi(s).$$

This potential-based shaping reward is added to the natural reward for every state transition the learner experiences. Call the augmented reward $r'_t = r_t + f(s_t, s_{t+1})$, and the value function based on this reward $V'(s)$. The potential-based shaping concept can also be applied to actions as well as states. See Wiewiora et al. (2003) for details.

It can be shown that the augmented value function is closely related to the original:

$$V'(s) = V(s) - \Phi(s).$$

An obvious choice for the potential function is $\Phi(s) \approx V(s)$, making $V'()$ close to zero. This intuition is strengthened by results presented by Wiewiora (2003). This paper shows that for most reinforcement learning systems, the potential function acts as an initial estimate of the natural value function $V()$.

However, even if the potential function used for shaping is very close to the true natural value function, learning may still be difficult. Koenig et al. have shown that initial estimates of the value function have a large influence on the efficiency of reinforcement learning (Koenig and Simmons 1996). With an initial estimate of the value function set below the optimal value, many reinforcement learning algorithms could require learning time exponential in the state and action space in order to find a highly rewarding state. On the other hand, in nonrandom environments, an optimistic initialization the value function creates learning time that is polynomial in the state-action space before a goal is found.

Cross-References

▸ Reinforcement Learning

Recommended Reading

Koenig S, Simmons RG (1996) The effect of representation and knowledge on goal directed exploration with reinforcement-learning algorithms. Mach Learn 22(1–3):227–250

Mataric MJ (1994) Reward functions for accelerated learning. In: International conference on machine learning, New Brunswick. Morgan Kaufmann, San Francisco, pp 181–189

Ng AY, Harada D, Russell S (1999) Policy invariance under reward transformations: theory and application to reward shaping. In: Machine learning, proceedings of the sixteenth international conference, Bled. Morgan Kaufmann, San Francisco, pp 278–287

Randlov J, Alstrom P (1998) Learning to drive a bicycle using reinforcement learning and shaping. In: Proceedings of the fifteenth international conference on machine learning, Madison. Morgan Kaufmann, San Francisco

Wiewiora E (2003) Potential-based shaping and Q-value initialization are equivalent. J Artif Intell Res 19: 205–208

Wiewiora E, Cottrell G, Elkan C (2003) Principled methods for advising reinforcement learning agents. In: Machine learning, proceedings of the twentieth international conference, Washington, DC. AAAI Press, Menlo Park, pp 792–799

Robot Learning

Jan Peters[1,2,3], Russ Tedrake[4], Nick Roy[4], and Jun Morimoto[5]
[1]Max Planck Institute for Biological Cybernetics, Tübingen, Germany
[2]Intelligent Autonomous Systems, Computer Science Department, Technische Universität Darmstadt, Darmstadt, Hessen, Germany
[3]Department of Empirical Inference, Max-Planck Institute for Intelligent Systems, Tübingen, Germany
[4]Massachusetts Institute of Technology, Cambridge, MA, USA
[5]Advanced Telecommunication Research Institute International (ATR), Kyoto, Japan

Definition

▸ Robot learning consists of a multitude of machine learning approaches, particularly ▸ reinforcement learning, ▸ inverse reinforcement learning, and ▸ regression methods, that have been adapted sufficiently to domain so that they allow learning in complex robot systems such as helicopters, flapping-wing flight, legged robots, anthropomorphic arms, and humanoid robots. While classical artificial intelligence-based robotics approaches have often attempted to manually generate a set of rules and models that allows the robot systems to sense and act in the real world, ▸ robot learning centers around the idea that it is unlikely that we can foresee all interesting real-world situations sufficiently accurate. Hence, the field of ▸ robot learning assumes that future robots need to be able to adapt to the real world, and domain-appropriate machine learning might offer the most approach in this direction.

Robot Learning Systems

As learning has found many backdoor entrances to robotics, this section can only scratch the surface. However, robot learning has clearly been successful in several areas: (i) model learning, (ii) imitation and apprenticeship learning, and (iii) reinforcement learning as well as in various other topics.

Model Learning

Model learning is the machine learning counterpart to classical system identification (Farrell and Polycarpou 2006; Schaal et al. 2002). However, while the classical approaches heavily rely on the structure of physically based models, specification of the relevant state variables, and hand-tuned approximations of unknown nonlinearities, model learning approaches avoid many of these labor-intensive steps and the entire process to be more easily automated. Machine learning and system identification approaches often assume an observable state of the system to estimate the mapping from inputs to outputs of the system. However, a learning system is often able to learn this mapping including the statistics needed to cope with unidentified state variables and can hence cope with a larger class of systems. Two types of models are commonly learned, i.e., forward and inverse models.

Forward models predict the behavior of the system based either on the current state or a history of preceding observations. They can be viewed as "learned simulators" that may be used for optimizing a policy or for predicting future information. Examples of the application of such learned simulators range from the early work in the late 1980s by Atkeson and Schaal in robot arm-based cart pole swing-ups to Ng's recent extensions for stabilizing an inverted helicopter. Most forward models can directly be learned by ▸ regression.

Conversely, *inverse models* attempt to predict the input to a system in order to achieve a desired output in the next step, i.e., it uses the model of the system to directly generate control signals. In traditional control, these are often called approximation-based control systems (Farrell and Polycarpou 2006). Inverse model learning can be straightforwardly by ▸ regression when the system dynamics can be inverted uniquely, e.g., as in inverse dynamic learning for a fully actuated system. However, for underactuated or redundantly actuated systems (Tedrake 2009), operational space control (Peters and Schaal 2008a), etc., such unique inverses do not exist and additional optimization is needed.

Imitation and Apprenticeship Learning

A key problem in robotics is to ease the problem of programming a complex behavior. Traditional robot programming approaches rely on accurate, manual modeling of the task and removal of all uncertainties so that they work well. In contrast to classical robot programming, learning from demonstration approaches aims at recovering the instructions directly from a human demonstration. Numerous unsolved problems exist in this context such as discovering the intent of the teacher or determining the mapping from the teacher's kinematics to the robot's kinematics (often called the correspondence problem). Two different approaches are common in this area, i.e., direct imitation learning and apprenticeship learning.

In *imitation learning* (Schaal et al. 2003), also known as ▸ behavioral cloning, the robot system directly estimates a policy from a teachers presentation, and, subsequently, the robot system reproduces the task using this policy. A key advantage of this approach is that it can often learn a task successfully from few demonstrations. In areas where human demonstrations are straightforward to obtain, e.g., for learning racket sports, manipulation, drumming on anthropomorphic systems, direct imitation learning often proved to be an appropriate approach. Its major shortcomings are that it cannot explain why the derived policy is a good one, and it may struggle with learning from noisy demonstrations.

Hence, *apprenticeship learning* (Coates et al. 2009) has been proposed as an alternative where a reward function is used as explanation of the teachers' behavior. Here, the reward function is chosen under which the teacher appears to act optimally, and the optimal policy for this reward function is subsequently computed as a solution. This approach transforms the problem of learning from demonstrations onto the harder problem of approximate optimal control or reinforcement learning; hence it is also known as inverse optimal control or ▸ inverse reinforcement learning. As a result, it is limited to problems that can be solved by current reinforcement learning methods. Additionally, it often has a hard time dealing with tasks where only few demonstrations with low variance exist. Hence, inverse reinforcement learning has been particularly successful in areas where it is hard for a human to demonstrate the desired behavior such as for helicopter acrobatics or in robot locomotion.

Further information on learning by demonstration may be found in Coates et al. (2009) and Schaal et al. (2003).

Robot Reinforcement Learning

The ability to self-improve with respect to an arbitrary reward function, i.e., ▸ reinforcement learning, is essential for robot systems to become more autonomous. Here, the system learns about its policy by interacting with its environment and receiving scores (i.e., rewards or costs) for the quality of its performance. Unlike supervised learning approaches used in model learning or imitation learning, reinforcement learning can still be considered to be in its infancy. Few off-the-shelf reinforcement learning methods scale into the domain of robotics both in terms of dimensionality and the number of trials needed to obtain an interesting behavior. Three different but overlapping styles of reinforcement learning can be found in robotics, i.e., model-based reinforcement learning, ▸ value function approximation methods, and direct ▸ policy search.

Model-based reinforcement learning relies upon a learned forward model used for simulation-based optimization as discussed before. While often highly efficient, it frequently suffers from the fact that learned models are imperfect, and hence, the optimization method can be guaranteed to be biased by the errors in the model. To date, a full Bayesian treatment of the model uncertainty appears to be a promising way for alleviating this shortcoming of this otherwise powerful approach.

Value function approximation methods have been the core approach used in reinforcement learning during the 1990s. These techniques rely upon approximating the expected rewards for every possible action in every visited state. Subsequently, the controller chooses the actions in accordance to this value. Such approximation requires a globally consistent value function where the quality of the policy is determined by the largest error of the value function at any possible state. As a result, these methods have been problematic for anthropomorphic robotics as the high-dimensional domains often defy learning such a global construct. However, it has been highly successful in low-dimensional domains such as mobile vehicle control and robot soccer, as well as on well-understood test domains such as cart-pole systems.

Unlike the previous two approaches, *policy search* attempts to directly learn the optimal policy from experience without solving intermediary learning problems. Policies often have significantly fewer parameters than models or value functions. For example, for balancing a ball on a plate (where the plate is mounted on a robot end effector) optimally with respect to a quadratic reward function, the number of policy parameters grows linearly in the number state dimensions, while it grows quadratically for both model and value function for this analytically tractable problem (in general cases, the number of parameters of value functions grows exponentially in the number of states which is known as the "curse of dimensionality"). This insight has given rise to policy search methods, particularly, ▸ policy gradient methods and probabilistic approaches to policy search such as the reward-weighted regression or PoWER. To date, application results of direct policy search approaches range from gait optimization in locomotion to various motor learning examples (e.g., Kendama, T-Ball, or throwing darts).

Further information on reinforcement learning for robotics may be found in Tedrake et al. (2004), Peters and Schaal (2008b), and Riedmiller et al. (2009).

Application Domains

The possible application domains for robot learning have not been fully explored, one could even aggressively state that we have barely started to bring learning into robotics. Nevertheless, robot learning has been successful in several application domains.

For accurate execution of desired trajectories, model learning has scaled to learning the full inverse dynamics for a humanoid robot in real time more accurately than achievable with physical models. Current work focusses mainly on improving the concurrent execution of tasks as well as control of redundant or underactuated systems.

Various approaches have been successful in task learning. Learning by demonstration approaches is moving increasingly toward industrial grade solutions where fast training of complex tasks becomes possible. Skills ranging from motor toys, e.g., basic movements, paddling a ball, etc., to complex tasks such as cooking a complete meal, basic table tennis strokes, helicopter acrobatics, or foot placement in locomotion have been learned from human teachers. Reinforcement learning has yielded better gaits in locomotion, jumping behaviors for legged robots, perching with fixed wing flight robots, forehands in table tennis, as well as various applications to learning of motor toys.

Cross-References

- ▸ Behavioral Cloning
- ▸ Inverse Reinforcement Learning
- ▸ Policy Search
- ▸ Reinforcement Learning
- ▸ Value Function Approximation

Recommended Reading

Coates A, Abbeel P, Ng AY (2009) Apprenticeship learning for helicopter control. Commun ACM 52(7):97–105

Farrell JA, Polycarpou MM (2006) Adaptive approximation based control. Adaptive and learning systems for signal processing, communications and control series. Wiley, Hoboken

Peters J, Schaal S (2008a) Learning to control in operational space. Int J Robot Res 27:197–212

Peters J, Schaal S (2008b) Reinforcement learning of motor skills with policy gradients. Neural Netw 21(4):682–697

Riedmiller M, Gabel T, Hafner R, Lange S (2009) Reinforcement learning for robot soccer. Auton Robot 27(1):55–73

Schaal S, Atkeson CG, Vijayakumar S (2002) Scalable techniques from nonparameteric statistics for real-time robot learning. Appl Intell 17(1):49–60

Schaal S, Ijspeert A, Billard A (2003) Computational approaches to motor learning by imitation. Philos Trans R Soc Lond: Ser B Biol Sci 358(1431): 537–547

Tedrake R (2009) Underactuated robotics: learning, planning, and control for efficient and agile machines. Course notes for MIT 6.832, MIT 32-380, Cambridge

Tedrake R, Zhang TW, Seung HS (2004) Stochastic policy gradient reinforcement learning on a simple 3d biped. In: Proceedings of the IEEE international conference on intelligent robots and systems (IROS), Sendai, pp 2849–2854

ROC Analysis

Peter A. Flach
Department of Computer Science, University of Bristol, Bristol, UK

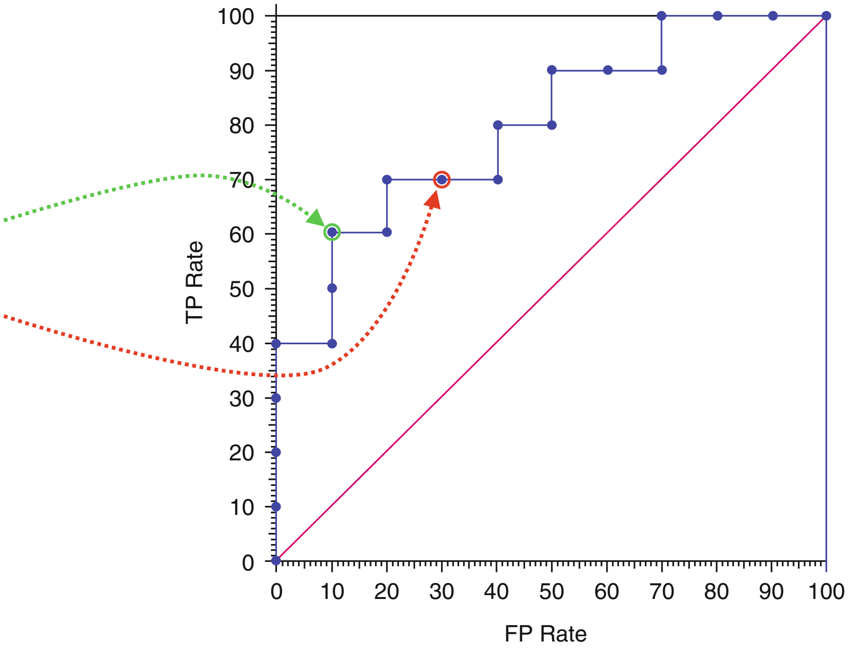
Synonyms

Receiver operating characteristic analysis

Definition

ROC analysis investigates and employs the relationship between ▸ sensitivity and ▸ specificity of a binary classifier. *Sensitivity* or ▸ *true positive rate* measures the proportion of positives correctly classified; *specificity* or ▸ *true negative rate* measures the proportion of negatives correctly classified. Conventionally, the true positive rate *tpr* is plotted against the ▸ *false positive rate fpr*, which is one minus true negative rate. If a classifier outputs a score proportional to its belief that an instance belongs to the positive class, decreasing the ▸ decision threshold – above which an instance is deemed to belong to the positive class – will increase both true

Class	Score
+	0.98
+	0.93
+	0.87
+	0.84
–	0.79
+	0.73
+	0.67
–	0.62
+	0.57
–	0.54
–	0.48
+	0.43
–	0.37
+	0.34
–	0.28
–	0.24
+	0.18
–	0.12
–	0.09
–	0.03



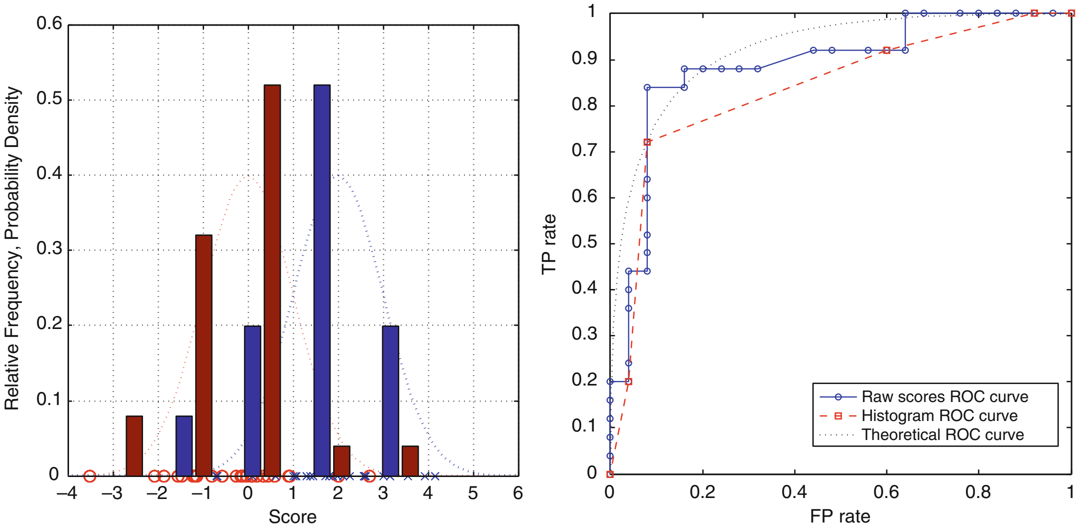
ROC Analysis, Fig. 1 The table on the *left* gives the scores assigned by a classifier to 10 positive and 10 negative examples. Each threshold on the classifier's score results in particular true and false positive rates, e.g., thresholding the score at 0.5 results in three misclassified positives ($tpr = 0.7$) and three misclassified negatives ($fpr = 0.3$); thresholding at 0.65 yields $tpr = 0.6$ and $fpr = 0.1$. Considering all possible thresholds gives the ROC curve on the *right*; this curve can also be constructed without explicit reference to scores, by going down the examples sorted on decreasing score and making a step up (to the *right*) if the example is positive (negative)

and false positive rates. Varying the decision threshold from its maximal to its minimal value results in a piecewise linear curve from $(0, 0)$ to $(1, 1)$, such that each segment has a nonnegative slope (Fig. 1). This *ROC curve* is the main tool used in ROC analysis. It can be used to address a range of problems, including: (1) determining a decision threshold that minimizes ▶ error rate or misclassification cost under given class and cost distributions; (2) identifying regions where one classifier outperforms another; (3) identifying regions where a classifier performs worse than chance; (4) obtaining calibrated estimates of the class posterior.

Motivation and Background

ROC analysis has its origins in *signal detection theory* (Egan 1975). In its simplest form, a detection problem involves determining the value of a binary signal contaminated with random noise. In the absence of any other information, the most sensible decision threshold would be halfway between the two signal values. If the noise distribution is zero centered and symmetric, sensitivity and specificity at this threshold have the same expected value, which means that the corresponding *operating point* on the ROC curve is located at the intersection with the descending diagonal $tpr + fpr = 1$. However, we may wish to choose different operating points, for instance, because false negatives and false positives have different costs. In that case, we need to estimate the noise distribution.

A slight reformulation of the signal detection scenario clarifies its relevance in a machine learning setting. Instead of superimposing random noise on a deterministic signal, we can view the resulting noisy signal as coming from a ▶ mixture distribution consisting of two compo-



ROC Analysis, Fig. 2 (*left*) Artificial classifier "scores" for two classes were obtained by sampling 25 points each from two ▶ Gaussian distributions with mean 0 and 2 and unit variance. The figure shows the raw scores on the x-axis and normalized histograms obtained by uniform five-bin discretization (*right*) The jagged ROC curve was obtained by thresholding the raw scores as before. The histogram gives rise to a smoothed ROC curve with only five segments. The *dotted line* is the theoretical curve obtained from the true Gaussian distributions

nent distributions with different means. The detection problem is now to decide, given a received value, from which component distribution it was drawn. This is essentially what happens in a binary ▶ classification scenario, where the scores assigned by a trained classifier follow a mixture distribution with one component for each class. The random variations in the data are translated by the classifier into random variations in the scores, and the classifier's performance depends on how well the per-class score distributions are separated. Figure 2 illustrates this for both discrete and continuous distributions. In practice, empirical ROC curves and distributions obtained from a test set are discrete because of the finite resolution supplied by the test set. This resolution is further reduced if the classifier only assigns a limited number of different scores, as is the case with ▶ decision trees; the histogram example illustrates this.

Solutions

For convenience, we will assume henceforth that score distributions are discrete and that decision thresholds always fall between actual scores (the results easily generalize to continuous distributions using probability density functions). There is a useful duality between thresholds and scores: decision thresholds correspond to operating points connecting two segments in the ROC curve, and actual scores correspond to segments of the ROC curve connecting two operating points. Let $f(s|+)$ and $f(s|-)$ denote the relative frequency of positive (negative) examples from a test set being assigned score s. (Note that s itself may be an estimate of the likelihood $p(x|+)$ of observing a positive example with feature vector x. We will return to this later.)

Properties of ROC Curves

The first property of note is that the true (false) positive rate achieved at a certain decision threshold t is the proportion of the positive (negative) score distribution to the right of the threshold; that is, $tpr(t) = \sum_{s>t} f(s|+)$ and $fpr(t) = \sum_{s>t} f(s|-)$. In Fig. 2, setting the threshold at 1 using the discretized scores gives a true positive rate of 0.72 and a false positive rate of 0.08, as can be seen by summing the bars of the histogram to the right of the threshold. Although the ROC curve does not display thresholds or scores, this allows us to reconstruct the range of thresholds yielding a particular operating point from the score distributions.

If we connect two distinct operating points on an ROC curve by a straight line, the slope of that line segment is equal to the ratio of positives to negatives in the corresponding score interval; that is,

$$slope(t_1, t_2) = \frac{tpr(t_2) - tpr(t_1)}{fpr(t_2) - fpr(t_1)} = \frac{\sum_{t_1<s<t_2} f(s|+)}{\sum_{t_1<s<t_2} f(s|-)}$$

Choosing the score interval small enough to cover a single segment of the ROC curve corresponding to score s, it follows that the segment has slope $f(s|+)/f(s|-)$. This can be verified in Fig. 2, e.g., the top-right segment of the smoothed curve has slope 0 because the leftmost bin of the histogram contains only negative examples. For continuous distributions, the slope of the ROC curve at any operating point is equal to the ratio of probability densities at that score.

It can happen that $slope(t_1, t_2) < slope(t_1, t_3) < slope(t_2, t_3)$ for $t_1 < t_2 < t_3$, which means that the ROC curve has a "dent" or *concavity*. This is inevitable when using raw classifier scores (unless the positives and negatives are perfectly separated), but can also be observed in the smoothed curve in the example: the rightmost bin of the histogram has a positive-to-negative ratio of 5, while the next bin has a ratio of 13. Consequently, the two leftmost segments of the ROC curve display a slight concavity. What this means is that performance can be improved by combining those two bins, leading to one large segment with slope 9. In other words, ROC curve concavities demonstrate locally suboptimal behavior of a classifier. An extreme case of suboptimal behavior occurs if the entire curve is concave or at least below the ascending diagonal: in that case, performance can simply be improved by assigning all test instances the same score, resulting in an ROC curve that follows the ascending diagonal. A *convex* ROC curve is one without concavities.

The AUC Statistic

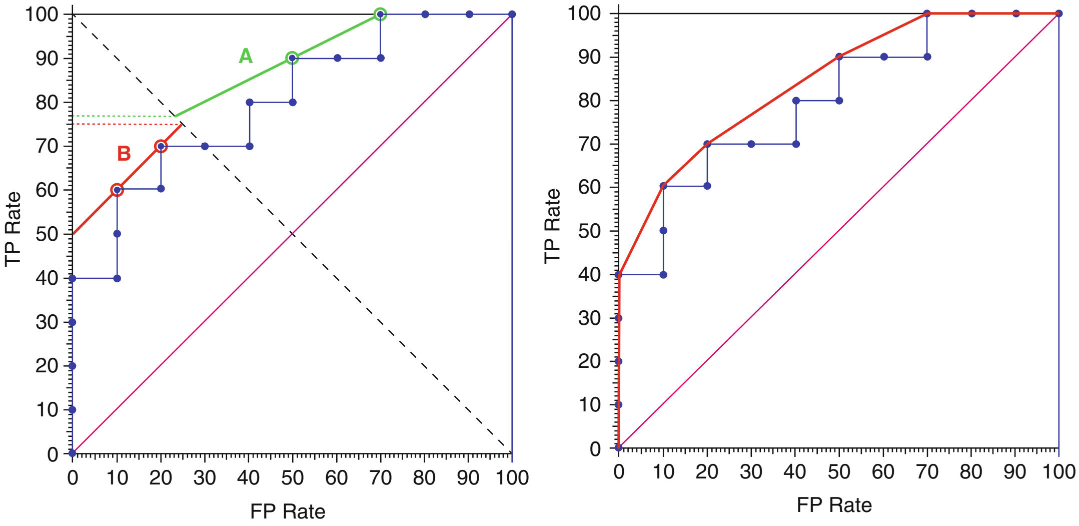
The most important statistic associated with ROC curves is the *area under (ROC) curve* or *AUC*. Since the curve is located in the unit square, we have $0 \leq AUC \leq 1$. $AUC = 1$ is achieved if the classifier scores every positive higher than every negative; $AUC = 0$ is achieved if every negative is scored higher than every positive. $AUC = 1/2$ is obtained in a range of different scenarios, including: (i) the classifier assigns the same score to all test examples, whether positive or negative, and thus the ROC curve is the ascending diagonal; (ii) the per-class score distributions are similar, which results in an ROC curve close (but not identical) to the ascending diagonal; and (iii) the classifier gives half of a particular class the highest scores and the other half the lowest scores. Notice that, although a classifier with *AUC* close to one half is often said to perform randomly, there is nothing random in the third classifier: rather, its excellent performance on some of the examples is counterbalanced by its very poor performance on some others (Sometimes a linear rescaling $2 \cdot AUC - 1$ called the *Gini coefficient* is preferred, which has a related use in the assessment of income or wealth distributions using Lorenz curves: a Gini coefficient close to 0 means that income is approximately evenly distributed. Notice that this Gini coefficient is often called the Gini index, but should not be confused with the impurity measure used in ▸ decision tree learning).

AUC has a very useful statistical interpretation: it is the expectation that a (uniformly) randomly drawn positive receives a higher score than a randomly drawn negative. It is a normalized version of the *Wilcoxon-Mann-Whitney sum of ranks test*, which tests the null hypothesis that two samples of ordinal measurements are drawn from a single distribution. The "sum of ranks" epithet refers to one method to compute this statistic, which is to assign each test example an integer rank according to decreasing score (the highest-scoring example gets rank 1, the next gets rank 2, etc.); sum up the ranks of the n^- negatives, which we want to be high; and subtract $\sum_{i=1}^{n^-} i = n^-(n^- + 1)/2$ to achieve 0 if all negatives are ranked first. The *AUC* statistic is then obtained by normalizing by the number of pairs of one positive and one negative, n^+n^-. There are several other ways to calculate *AUC*, for instance, we can calculate, for each negative, how many positives precede it, which basically is a columnwise calculation and yields an alternative view of *AUC* as the expected true positive rate if the operating point is chosen just before a randomly drawn negative.

Identifying Optimal Points and the ROC Convex Hull

In order to select an operating point on an ROC curve, we first need to specify the objective function we aim to optimize. In the simplest case, this will be ▶ accuracy, the proportion of correctly predicted examples. Denoting the proportion of positives by *pos*, we can express accuracy as a weighted average of the true positive and true negative rates $pos \cdot tpr + (1 - pos)(1 - fpr)$. It follows that points with the same accuracy lie on a straight line with slope $a = (1 - pos)/pos$; these parallel lines are the *isometrics* for accuracy (Peter 2003). In order to find the optimal operating point for a given class distribution, we can start with an accuracy isometric through $(0, 1)$ and slide it down until it touches the ROC curve in one or more points (Fig. 3 (left)). In the case of a single point, this uniquely determines the operating point and thus the threshold. If there are several points in common between the accuracy isometric and the ROC curve, we can make an arbitrary choice or interpolate stochastically. We can read off the achieved accuracy by intersecting the accuracy isometric with the descending diagonal, on which $tpr = 1 - fpr$, and therefore the true positive rate at the intersection point is equal to the accuracy associated with the isometric.

We can generalize this approach to any objective function that is a linear combination of true and false positive rates. For instance, let predicting class i for an instance of class j



ROC Analysis, Fig. 3 (*left*) The slope of accuracy isometrics reflects the class ratio. Isometric A has slope 1/2: this corresponds to having twice as many positives as negatives, meaning that an increase in true positive rate of x is worth a $2x$ increase in false positive rate. This selects two optimal points on the ROC curve. Isometric B corresponds to a uniform class distribution and selects optimal points which make fewer positive predictions. In either case, the achieved accuracy can be read off on the y-axis after intersecting the isometric with the descending diagonal (slightly higher for points selected by A). (*right*) The convex hull selects those points on an ROC curve which are optimal under some class distribution. The slope of each segment of the convex hull gives the class ratio under which the two end points of the segment yield equal accuracy. All points under the convex hull are non-optimal incur cost $cost(i|j)$, so, for instance, the cost of a false positive is $cost(+|-)$ (profits for correct predictions are modeled as negative costs, e.g., $cost(+|+) < 0$). Cost isometrics then have slope

$$\frac{cost(+|-) \; - \; cost(-|-)}{cost(-|+) \; - \; cost(+|+)}$$

Nonuniform class distributions are simply taken into account by multiplying the class and cost ratio, giving a single *skew ratio* expressing the relative importance of negatives compared to positives.

This procedure of selecting an optimal point on an ROC curve can be generalized to select among points lying on more than one curve or even an arbitrary set of points (e.g., points representing different categorical classifiers). In such scenarios, it is likely that certain points are never selected for any skew ratio; such points are said to be *dominated*. For instance, points on a concave region of an ROC curve are dominated. The nondominated points are optimal for a given closed interval of skew ratios and can be joined to form the *convex hull* of the given ROC curve or set of ROC points (Fig. 3 (right)); in multi-objective optimization, this concept is called the *Pareto front*. This notion of the ROC convex hull (sometimes abbreviated to ROCCH) is extremely useful in a range of situations. For instance, if an ROC curve displays concavities, the convex hull represents a discretization of the scores which achieves higher AUC. Alternatively, the convex hull of a set of categorical classifiers can be interpreted as a hybrid classifier that can reach any point on the convex hull by stochastic interpolation between two neighboring classifiers (Foster and Tom 2001).

Obtaining Calibrated Estimates of the Class Posterior

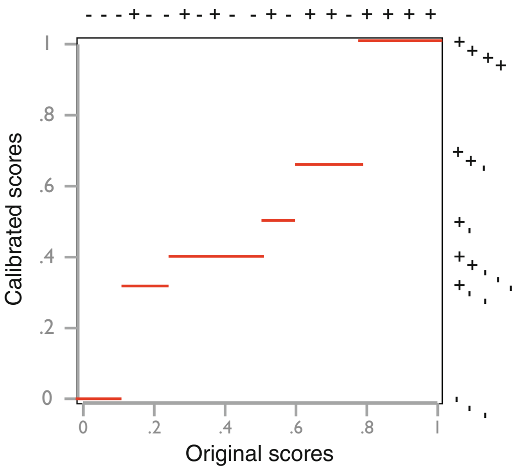
Recall that each segment of an ROC curve has slope $slope(s) = f(s|+)/f(s|-)$, where s is the score associated with the segment, and $f(s|+)$ and $f(s|-)$ are the relative frequencies of positives and negatives assigned score s. Now consider the function

$$\begin{aligned} cal(s) &= \frac{pos \cdot f(s|+)}{pos \cdot f(s|+) \; + \; (1-pos) \cdot f(s|-)} \\ &= \frac{slope(s)}{slope(s) + a} \end{aligned}$$

with $a = (1-pos)/pos$. The *calibration map* $s \mapsto cal(s)$ adjusts the classifier's scores to reflect the empirical probabilities observed in the test set. If the ROC curve is convex, $slope(s)$ and $cal(s)$ are monotonically nonincreasing with decreasing s, and thus replacing the scores s with $cal(s)$ does not change the ROC curve (other than merging neighboring segments with different scores but the same slope into a single segment).

Consider ▸ decision trees as a concrete example. Once we have trained (and possibly pruned) a tree, we can obtain a score in each leaf l by taking the proportion of positive training examples in that leaf: $score(l) = p(+|l)/(p(+|l) + p(-|l))$. Each leaf of the tree then gives rise to a different segment of the ROC curve, which, by the nature of how the scores were calculated, will be convex. Furthermore, we have that $cal(score(l)) = score(l)$, which means that the tree produces posterior probabilities that are perfectly calibrated with respect to the training set. If we anticipate changes in class distribution, we may choose to calibrate with a different a. For example, if we use $a = 1$, the calibrated scores $cal(score(l))$ are adjusted for a uniform prior.

If the ROC curve is not convex, the mapping $s \mapsto cal(s)$ is not monotonic; while the scores $cal(s)$ would lead to improved performance on the data from which the ROC curve was derived, this is very unlikely to generalize to other data and thus leads to ▸ overfitting. This is why, in practice, a less drastic calibration procedure involving the convex hull is applied (Tom and Alexandru 2007). Let s_1 and s_2 be the scores associated with the start and end segments of a concavity, i.e., $s_1 > s_2$ and $slope(s_1) < slope(s_2)$. Let $slope(s_1 s_2)$ denote the slope of the line segment of the convex hull that repairs this concavity, which implies $slope(s_1) < slope(s_1 s_2) < slope(s_2)$. The calibration map will then map any score in the interval $[s_1, s_2]$ to $slope(s_1 s_2)/(slope(s_1 s_2) + 1)$ (Fig. 4).



ROC Analysis, Fig. 4 The piecewise constant calibration map derived from the convex hull in Fig. 3. The original score distributions are indicated at the *top* of the figure, and the calibrated distributions are on the *right*. We can clearly see the combined effect of binning the scores and redistributing them over the interval [0, 1]

This ROC-based calibration procedure, which is also known as *isotonic regression* (Barbara and Charles 2002), not only produces calibrated probability estimates but also improves AUC. This is in contrast with other calibration procedures such as logistic calibration which do not bin the scores and therefore do not change the ROC curve. ROC-based calibration can be shown to achieve the lowest *Brier score* (Glenn 1950), which measures the mean squared error in the probability estimates as compared with the ideal probabilities (1 for a positive and 0 for a negative), among all probability estimators that do not reverse pairwise rankings. On the other hand, being a nonparametric method, it typically requires more data than parametric methods in order to estimate the bin boundaries reliably. See ▶ Classifier Calibration for further details.

Future Directions

ROC analysis in its original form is restricted to binary ▶ classification, and its extension to more than two classes gives rise to many open problems. c-class ROC analysis requires $c(c-1)$ dimensions, in order to distinguish each possible misclassification type. Srinivasan proved that basic concepts such as the ROC polytope and its linearly interpolated convex hull generalize to the c-class case (Ashwin 1999). In theory, the volume under the ROC polytope can be employed for assessing the quality of a multi-class classifier (César et al. 2003), but this volume is hard to compute as – unlike the two-class case, where the segments of an ROC curve can simply be enumerated in $O(n \log n)$ time by sorting the n examples on their score (Tom 2006; Peter 2004) – there is no simple way to enumerate the ROC polytope. Mossman considers the special case of three-class ROC analysis, where for each class the two possible misclassifications are treated equally (a so-called one-versus-rest scenario) (Douglas 1999). Hand and Till propose the average of all one-versus-rest AUCs as an approximation of the area under the ROC polytope (David and Robert 2001). Various algorithms for minimizing a classifier's misclassification costs by reweighting the classes are considered in Nicolas and Peter (2003) and Chris et al. (2008).

Other research directions include the explicit visualization of misclassification costs (Chris and Robert 2006) and using ROC analysis to study the behavior of machine learning algorithms and the relations between machine learning metrics (Johannes and Peter 2005).

Cross-References

▶ Accuracy
▶ Classification
▶ Classifier Calibration
▶ Confusion Matrix
▶ Cost-Sensitive Learning
▶ Error Rate
▶ False Negative
▶ False Positive
▶ Gaussian Distribution
▶ Posterior Probability
▶ Precision
▶ Prior Probability
▶ Recall

- ▶ Sensitivity
- ▶ Specificity
- ▶ True Negative
- ▶ True Positive

Recommended Reading

Bourke C, Deng K, Scott S, Schapire R, Vinodchandran NV (2008) On reoptimizing multi-class classifiers. Mach Learn 71(2–3):219–242

Brier G (1950) Verification of forecasts expressed in terms of probabilities. Mon Weather Rev 78:1–3

Drummond C, Holte R (2006) Cost curves: an improved method for visualizing classifier performance. Mach Learn 65(1):95–130

Egan J (1975) Signal detection theory and ROC analysis. Series in cognitition and perception. Academic Press, New York

Fawcett T (2006) An introduction to ROC analysis. Patt Recognit Lett 27(8):861–874

Fawcett T, Niculescu-Mizil A (2007) PAV and the ROC convex hull. Mach Learn 68(1):97–106

Ferri C, Hernández-Orallo J, Salido M (2003) Volume under the ROC surface for multi-class problems. In: Proceedings of the fourteenth European conference on machine learning, Cavtat, pp 108–120

Flach P (2003) The geometry of ROC space: understanding machine learning metrics through ROC isometrics. In: Proceedings of the twentieth international conference on machine learning (ICML 2003), Washington, DC, pp 194–201

Flach P (2004) The many faces of ROC analysis in machine learning, July 2004. ICML-04 Tutorial. Notes available from http://www.cs.bris.ac.uk/~flach/ICML04tutorial/index.html

Fuernkranz J, Flach P (2005) ROC 'n' Rule learning – towards a better understanding of covering algorithms. Mach Learn 58(1):39–77

Hand D, Till R (2001) A simple generalization of the area under the ROC curve to multiple class classification problems. Mach Learn 45(2):171–186

Lachiche N, Flach P (2003) Improving accuracy and cost of two-class and multi-class probabilistic classifiers using ROC curves. In: Proceedings of the twentieth international conference on machine learning (ICML'03), Washington, DC, pp 416–423

Mossman D (1999) Three-way ROCs. Med Decis Mak 19:78–89

Provost F, Fawcett T (2001) Robust classification for imprecise environments. Mach Learn 42(3):203–231

Srinivasan A (1999) Note on the location of optimal classifiers in n-dimensional ROC space. Technical report PRG-TR-2-99, Oxford University Computing Laboratory, Oxford

Zadrozny B, Elkan C (2002) Transforming classifier scores into accurate multiclass probability estimates. In: Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining, Edmonton. ACM, pp 694–699

ROC Convex Hull

The convex hull of an ▶ ROC curve is a geometric construction that selects the points on the curve that are optimal under some class and cost distribution. It is analogous to the Pareto front in multiobjective optimization. See ▶ ROC Analysis.

ROC Curve

The ROC curve is a plot depicting the trade-off between the ▶ true positive rate and the ▶ false positive rate for a classifier under varying decision thresholds. See ▶ ROC Analysis.

Rotation Forests

Rotation Forests is an ▶ ensemble learning technique. It is similar to the ▶ Random Forests approach to building decision tree ensembles. In the first step, the original feature set is split randomly into K disjoint subsets. Next, ▶ principal components analysis is used to extract n principal component dimensions from each of the K subsets. These are then pooled, and the original data projected linearly into this new feature space. A tree is then built from this data in the usual manner. This process is repeated to create an ensemble of trees, each time with a different random split of the original feature set.

As the tree learning algorithm builds the classification regions using hyperplanes parallel to the feature axes, a small rotation of the axes may lead to a very different tree. The effect of rotating the axes is that classification regions of high accuracy can be constructed with far fewer trees than in ▶ Bagging and ▶ Adaboost.

RSM

▸ Random Subspace Method

Rule Learning

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt,
Darmstadt, Deutschland
Department of Information Technology,
University of Leoben, Leoben, Austria

Abstract

Informally, rule learning denotes all algorithms that learn or discover patterns in data, which are formulated in the form of a ▸ rule. These can be predictive (e.g., ▸ classification rules) or descriptive rules (e.g., ▸ association rules or ▸ supervised descriptive rule induction). Consequently, the learning algorithms typically differ in the type of search they use for finding these rules in the search space. Exhaustive search is more common in descriptive rule mining, whereas heuristic search using a variety of quality criteria is more commonly used in predictive rule learning. An overview of the field can be found in Fürnkranz et al. (2012).

Learning Individual Rules

Conceptually, rule learning may be viewed as a search in the space of possible ▸ rules. The first algorithms, such as the candidate elimination algorithm, aimed at identifying the ▸ version space of all complete and consistent rules (Mitchell 1982). ▸ Association rule discovery algorithms look for all rules that satisfy certain constraints, typically all rules with a minimum coverage and a minimum support. Most flexible are algorithms that use heuristic search for optimizing given quality criteria. Such algorithms are also often used in ▸ supervised descriptive rule induction.

```
procedure FindBestRule(Examples,BestRule)

Input: Examples, a set of positive and negative examples
for a class c.

InitRule = InitializeRule(Examples)
InitVal = EvaluateRule(InitRule)
BestRule = <InitVal,InitRule>
Rules = {BestRule}
while Rules ≠ ∅ do
    Candidates = SelectCandidates(Rules,
    Examples)
    Rules = Rules \ Candidates
    for Candidate ∈ Candidates do
        Refinements = RefineRule(Candidate,
        Examples)
        for Refinement ∈ Refinements do
            Evaluation = EvaluateRule(Refinement,
            Examples)
            if StoppingCriterion(Refinement,
              Examples)
                then next Refinement
            NewRule = <Evaluation,Refinement>
            Rules = InsertSort(NewRule, Rules)
            if NewRule > BestRule
            then BestRule = NewRule
        endfor
    endfor
    Rules = FilterRules(Rules, Examples)
endwhile

Output: BestRule
```

FindBestRule is a prototypical algorithm that searches for a rule which optimizes a given quality criterion defined in EvaluateRule. The value of this heuristic function is the higher the more positive and the less negative examples are covered by the candidate rule. FindBestRule maintains *Rules*, a sorted list of candidate rules, which is initialized by the procedure InitializeRule. New rules will be inserted in appropriate places (InsertSort), so that *Rules* will always be sorted in decreasing order of the heuristic evaluations of the rules. At each cycle, SelectCandidates selects a subset of these candidate rules, which are then refined using the refinement operator RefineRule. Each refinement is evaluated and inserted into the sorted *Rules* list unless the StoppingCriterion prevents this. If the evaluation of the *NewRule* is better than the best rule found previously, *BestRule* is set to *NewRule*. FilterRules selects the subset of the ordered rule list that will be used in subsequent iterations. When all candidate rules have been processed, the best rule will be returned.

Different choices of these functions allow the definition of different biases for the separate-and-conquer learner. The search bias is defined by the choice of a search strategy (InitializeRule and RefineRule), a search algorithm (SelectCandidates and FilterRules), and a search heuristic (EvaluateRule). The refinement operator RefineRule constitutes the language bias of the algorithm. An overfitting avoidance bias can be implemented via some StoppingCriterion and/or in a post-processing phase.

For example, InitializeRule and RefineRule may be defined so that they realize a top-down (general-to-specific), a bottom-up (specific-to-general), or a bidirectional search. Exhaustive breadth-first, depth-first, or best-first searches can be realized by appropriate choices of EvaluateRule and no filtering or candidate selection. FilterRules can, e.g., be used to realize a hill-climbing or ▸ beam search by maintaining only the best or the *BeamWidth* best rules. Evolutionary algorithms and stochastic local search can also be easily realized.

The most common algorithm for finding the best rule is a top-down hill-climbing algorithm. It basically constructs a rule by consecutively adding conditions to the rule body so that a given quality criterion is greedily optimized. This constitutes a simple greedy hill-climbing algorithm for finding a local optimum in the hypothesis space defined by the feature set. InitializeRule will thus return the most general rule, the rule with the body {`true`}, and RefineRule will return all possible extensions of the rule by a single condition. FilterRules will only let the best refinement pass for the next iteration, so that SelectCandidates will always have only one choice. The search heuristic, the StoppingCriterion, and the post-processing are discussed in the next sections.

Rule Learning Heuristics

The goal of rule learning is to find a rule or a ▸ rule set that is as *complete* and *consistent* as possible. Thus, each rule should cover as many positive examples and as few negative examples as possible. A few important ones are (assume that p out of P positive examples and n out of N negative examples are covered by the rule):

Laplace estimate (Lap $= \frac{p+1}{p+n+2}$) computes the fraction of positive examples in all covered examples, where each class is initialized with one virtual example in order to penalize rules with low coverage.

***m*-Estimate** (m $= \frac{p+m\cdot P/(P+N)}{p+n+m}$) is a generalization of the Laplace estimate which uses m examples for initialization, which are distributed according to the class distribution in the training set (Cestnik 1990).

Information gain (ig $= p\cdot(\log_2 \frac{p}{p+n} - \log_2 \frac{p'}{p'+n'})$, where p' and n' are the number of positive and negative examples covered by the rule's predecessor) is Quinlan's (1990) adaptation of the information gain heuristic used for decision tree learning. The main difference is that this only focuses on a single branch (a rule), whereas the decision tree version tries to optimize all branches simultaneously.

correlation and χ^2 (corr$=\frac{p(N-n)-(P-p)n}{\sqrt{PN(p+n)(P-p+N-n)}}$) computes the four-field correlation of covered/uncovered positive/negative examples. It is equivalent to a χ^2 statistic ($\chi^2 = (P+N)\,\text{corr}^2$).

An exhaustive overview and theoretical comparison of various search heuristics in coverage space, a variant of ROC space can be found in Fürnkranz and Flach (2005).

Overfitting Avoidance

It is trivial to find a rule set that is complete and consistent on the training data. To achieve this, one only needs to convert each positive example into a rule. Each of these rules is consistent (provided the data set is not inconsistent), and collectively they cover the entire example set (completeness). However, this is clearly a bad case of ▸ overfitting because the theory will not generalize to new positive examples.

Overfitting is to some extent handled by the search heuristics described above, but most algorithms use additional ▸ pruning techniques. One can discriminate between *pre-pruning* techniques, where a separate criterion is used to filter out unpromising rules. For example, CN2 computes the *likelihood ratio statistic* lrs $= 2 \cdot (p \log \frac{p}{e_p} + n \log \frac{n}{e_n})$, where $e_p = (p + n)\frac{P}{P+N}$ and $e_n = (p+n)\frac{N}{P+N} = (p+n) - e_p$ are the number of positive and negative examples one could expect if the $p + n$ examples covered by the rule were distributed in the same way as the $P + N$ examples in the full data set. This statistic follows a χ^2 distribution, which allows to filter out rules for which the distribution of the covered examples is not statistically significantly different from the distribution of examples in the full data set. Other pre-pruning criteria are simple thresholds that define a minimum acceptable value for the search heuristic or FOIL's ▸ minimum description length criterion that relates the length of a rule to the number of examples it covers.

However, it can be shown experimentally that CN2 or FOIL still has a tendency to overfit the data. Instead, state-of-the-art algorithms post-prune a rule right after it has been learned. For this purpose, one-third of the training data are reserved for pruning. After a rule has been learned, its accuracy is greedily simplified on the pruning set. Simplifications can be the deletion of the last condition, a final sequence of conditions, or an arbitrary condition of the rule. If the simplification does not decrease the accuracy of the rule on the pruning set, it will be performed. This so-called *incremental reduced error pruning* algorithm (Fürnkranz and Widmer 1994) is used in the rule learning algorithm RIPPER.

A survey and experimental comparison of pruning techniques for rule learning can be found in Fürnkranz (1997).

Learning Rule Sets

In many cases, rule learning is used for solving a ▸ classification problem via the induction of a ▸ rule set or a ▸ decision list. In these cases, individual rules are learned as above but then combined to form a theory that is able to classify all examples. The principal approach is the so-called ▸ covering or ▸ separate-and-conquer algorithm, which learns one rule at a time, successively removing the covered examples. Individual algorithms within this framework differ primarily in the way they learn single rules.

An obvious generalization of covering is to not entirely remove covered examples but to reduce their example ▸ weights, thus decreasing their importance in subsequent iterations (see, e.g., the SLIPPER algorithm (Cohen and Singer 1999)).

Rules can also be learned by alternative strategies. There have been numerous proposals, and we can only mention the most influential. Each path from the root to a leaf of a ▸ decision tree corresponds to a rule and so rules can be learned by first learning a decision tree and then post-processing it (see, e.g., the C4.5RULES algorithm, (Quinlan 1993)). It is also possible to use the ▸ Apriori algorithm for an exhaustive search for classification rules and to use a subsequent covering algorithm to combine the rules into a rule set (see, e.g., the CBA algorithm (Liu et al. 1998)). RISE (Domingos 1996) combines bottom-up generalization with ▸ nearest neighbor algorithms to learn a theory via "conquering without separating."

Well-Known Rule Learning Algorithms

AQ can be considered as the original covering algorithm. Its original version was conceived by Ryszard Michalski in the 1960s (Michalski 1969), and numerous versions and variants of the algorithm appeared subsequently in the literature. AQ uses a top-down beam search for finding the best rule. It does not search all possible specializations of a rule but only considers refinements that cover a particular example, the so-called *seed example*. This idea is basically the same as the use of a ▸ bottom clause in ▸ inductive logic programming.

CN2 (Clark and Niblett 1989; Clark and Boswell 1991) employs a beam search guided by the Laplace or m-estimates, and the abovementioned likelihood ratio significance test to fight overfitting. It can operate in two modes, one for learning ▸ rule sets (by modeling each class independently) and one for learning ▸ decision lists.

FOIL (Quinlan 1990) was the first relational learning algorithm that received attention beyond the field of ▸ inductive logic programming. It learns a concept with the covering loop and learns individual concepts with a top-down refinement operator, guided by information gain. The main difference to previous systems is that FOIL allowed the use of first-order background knowledge. Instead of only being able to use tests on single attributes, FOIL could employ tests that compute relations between multiple attributes and could also introduce new variables in the body of a rule.

RIPPER was the first rule learning system that effectively countered the overfitting problem via *incremental reduced error pruning*, as described above. It also added a post-processing phase for optimizing a rule set in the context of other rules. The key idea is to remove one rule out of a previously learned rule set and try to relearn it not only in the context of previous rules (as would be the case in the regular covering rule) but also in the context of subsequent rules. RIPPER is still state of the art in inductive rule learning. A freely accessible re-implementation can be found in the WEKA machine learning library under the name of JRIP.

OPUS (Webb 1995) was the first rule learning algorithm to demonstrate the feasibility of a full exhaustive search through all possible rule bodies for finding a rule that maximizes a given quality criterion (or heuristic function). The key idea is the use of *ordered search* that prevents that a rule is generated multiple times. This means that even though there are $l!$ different orders of the conditions of a rule of length l, only one of them can be taken by the learner for finding this rule. In addition, OPUS uses several techniques that prune significant parts of the search space, so that this search method becomes feasible. Follow-up work has shown that this technique is also an efficient alternative for ▸ association rule discovery, provided that the database to mine fits into the memory of the learning system.

CBA was one of the first and best-known algorithms that employed association rule learning algorithms for learning predictive rules (Liu et al. 1998). In its simplest version, the algorithm selects the final rule sets by sorting all class association rules according to confidence and incrementally adding rules to the final set until all examples are covered or the quality of the rule set decreases.

Cross-References

▸ Association Rule
▸ Classification Rule
▸ Covering Algorithm
▸ Decision List
▸ Decision Lists and Decision Trees
▸ Rule Set
▸ Supervised Descriptive Rule Induction

Recommended Reading

Cestnik B (1990) Estimating probabilities: a crucial task in machine learning. In: Aiello L (ed) Proceedings of the 9th European conference on artificial intelligence (ECAI-90). Pitman, Stockholm, pp 147–150

Clark P, Boswell R (1991) Rule induction with CN2: some recent improvements. In: Proceedings of the 5th European working session on learning (EWSL-91). Springer, Porto, pp 151–163

Clark P, Niblett T (1989) The CN2 induction algorithm. Mach Learn 3(4):261–283

Cohen WW, Singer Y (1999) A simple, fast, and effective rule learner. In: Proceedings of the 16th national conference on artificial intelligence (AAAI-99). AAAI/MIT Press, Menlo Park, pp 335–342

Domingos P (1996) Unifying instance-based and rule-based induction. Mach Learn 24:141–168

Fürnkranz J (1997) Pruning algorithms for rule learning. Mach Learn 27(2):139–171. http://www.ke.informatik.tu-darmstadt.de/ juffi/publications/mlj97.pdf

Fürnkranz J, Flach PA (2005) ROC 'n' rule learning – towards a better understanding of covering algorithms. Mach Learn 58(1):39–77. doi:10.1007/s10994-005-5011-x. http://www.cs.bris.ac.uk/~flach/papers/furnkranz-flach-mlj.pdf

Fürnkranz J, Widmer G (1994) Incremental reduced error pruning. In: Cohen WW, Hirsh H (eds) Proceedings of the 11th international conference on machine learning (ML-94). Morgan Kaufmann, New Brunswick, pp 70–77. http://www.ke.informatik.tu-darmstadt.de/~juffi/publications/ml-94.ps.gz

Fürnkranz J, Gamberger D, Lavrač N (2012) Foundations of rule learning. Springer. doi:10.1007/978-3-540-75197-7. ISBN 978-3-540-75196-0. http://www.springer.com/978-3-540-75196-0

Liu B, Hsu W, Ma Y (1998) Integrating classification and association rule mining. In: Agrawal R, Stolorz P, Piatetsky-Shapiro G (eds) Proceedings of the 4th international conference on knowledge discovery and data mining (KDD-98), New York, pp 80–86

Michalski RS (1996) On the quasi-minimal solution of the covering problem. In: Proceedings of the 5th international symposium on information processing (FCIP-69), vol A3 (Switching circuits), Bled, pp 125–128

Mitchell TM (1982) Generalization as search. Artif Intell 18(2):203–226

Quinlan JR (1990) Learning logical definitions from relations. Mach Learn 5:239–266

Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, San Mateo

Webb GI (1995) OPUS: an efficient admissible algorithm for unordered search. J Artif Intell Res 5: 431–465

Rule Set

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt,
Darmstadt, Deutschland
Department of Information Technology,
University of Leoben, Leoben, Austria

Abstract

A rule set is a collection of individual ▶ classification rules that collectively form a classifier. In contrast to a ▶ decision list, the rules in the set do not have an inherent order, and all rules in the set have to be tried for deriving a prediction for an example.

Discussion

This may cause two types of problems that have to be resolved with additional algorithms:

Multiple rules fire: More than one rule can fire on a single example, and these rules can make contradicting predictions. This type of conflict is typically resolved by preferring rules that cover a higher fraction of training examples of their class (typically estimated with Laplace correction, see ▶ rule learning). This is equivalent to converting the rule set into a decision list that is ordered according to this evaluation heuristic. More elaborate tie-breaking schemes, such as using a *Naive Bayes* algorithm, or inducing a separate rule set for handling these conflicts (*double induction* (Lindgren and Boström 2004)), have also been tried.

No rule fires: It may also occur that no rule fires for a given example. Such cases are typically handled via a so-called *default rule*, which typically predicts the majority class. Again, a more complex algorithm, such as trying to find the closest rule (*rule stretching* (Eineborg and Boström 2001)), has been proposed.

A rule set that only contains rules for a single class, as is the result of ▶ concept learning problems, typically contains an implicit default rule for the other class (very much like a Prolog program). If all rules are conjunctive, such rule sets may be interpreted as a definition in *disjunctive normal form* for this class.

Cross-References

▶ Classification Rule
▶ Decision List
▶ Disjunctive Normal Form
▶ Rule Learning

Recommended Reading

Eineborg M, Boström H (2001) Classifying uncovered examples by rule stretching. In: Rouveirol C, Sebag M (eds) Proceedings of the eleventh international conference on inductive logic programming (ILP-01), Strasbourg. Springer, pp 41–50

Lindgren T, Boström H (2004) Resolving rule conflicts with double induction. Intell Data Anal 8(5): 457–468

S

Sample Complexity

▶ Generalization Bounds

Samuel's Checkers Player

Definition

Samuel's Checkers Player is the first machine learning system that received public recognition. It pioneered many important ideas in game playing and machine learning. The two main papers describing his research (Samuel 1959, 1967) became landmark papers in Artificial Intelligence. In one game, the resulting program was able to beat one of America's best players of the time.

Description of the Learning System

Samuel's checkers player featured a wide variety of learning techniques. First, his checkers player remembered positions that it frequently encountered during play. This simple form of *rote learning* allowed it to save time, and to search deeper in subsequent games whenever a stored position was encountered on the board or in some line of calculation. Next, it featured the first successful application of what is now known as ▶ Reinforcement Learning for tuning the weights of its evaluation function. The program trained itself by playing against a stable copy of itself. After each move, the weights of the evaluation function were adjusted in a way that moved the evaluation of the root position after a quiescence search closer to the evaluation of the root position after searching several moves deep. This technique is a variant of what is nowadays known as Temporal-Difference Learning and commonly used in successful game-playing programs. Samuel's program not only tuned the weights of the evaluation but also employed on-line ▶ Feature Selection for constructing the evaluation function with the terms that seem to be the most significant for evaluating the current board situation. ▶ Feature Construction was recognized as the key problem that still needs to be solved. Later, Samuel changed his evaluation function from a linear combination of terms into a structure that closely resembled a 3-layer ▶ Neural Network. This structure was trained with ▶ Preference Learning from several thousand positions from master games.

Cross-References

▶ Machine Learning and Game Playing

Recommended Reading

Samuel AL (1959) Some studies in machine learning using the game of checkers. IBM J Res Develop 3(3):211–229

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

Samuel AL (1967) Some studies in machine learning using the game of checkers. II – recent progress. IBM J Res Develop 11(6):601–617

Saturation

▸ Bottom Clause

SDP

▸ Symbolic Dynamic Programming

SDRI

▸ Supervised Descriptive Rule Induction

Search Engines: Applications of ML

Eric Martin
University of New South Wales, Sydney, NSW, Australia

Abstract

The general structure of a search engine is described. An overview of those information retrieval methods that are relevant to web search in that they take the existence of hyperlinks between documents into account, is provided. A suggested classification of web queries as either navigational, transactional, or informational has been suggested. More generally, a good understanding of users' needs and practice allows for query rewriting or for redirection to domain-specific databases.

Definition

Search engines provide users with Internet resources—links to websites, documents, text snippets, images, videos, . . .—in response to queries. They use techniques that are part of the field of information retrieval and rely on statistical and pattern matching methods. Search engines have to take into account many key aspects and requirements of this specific instance of the information retrieval problem. First is the fact that they have to be able to process hundreds of millions of searches a day and answer queries in a matter of milliseconds. Second is the fact that the resources on the World Wide Web are constantly updated, with information being continuously added, removed, or changed—the overall contents changing by up to 8 % a week—in a pool consisting of billions of documents. Third is the fact that users will express possibly semantically complex queries in a language with limited expressive power and often not make use or proper use of available syntactic features of that language—for instance, the Boolean *or* operator occurs in less than 3 % of queries.

Motivation and Background

Web searching is technically initiated by sending a query to a search engine, but the whole search process starts earlier, in the mind of the person who conducts the search. To be successful, the process needs to provide users with words, text snippets, images, or movies that fulfill the users' quest for information. So even though a search is technically the implementation of a procedure that maps a query to some digital material, it spans a larger spectrum of activities, from a psychological trigger to a psychological reward. For a given set of digital material that, if provided, would be deemed perfectly satisfactory by a number of users looking for the same information, different users will issue different queries. That might be because they have varying skills at conveying what they are after in the form of a few words. That might be because their understanding of the technology prompts them to formulate what they are after in a form that, rightly or wrongly, they consider appropriate for a computing device to process. That might be for a number of different reasons that all point to the fact that the quality of the search is not determined by its adequacy to the query, but by its adequacy to the psychological trigger that produced the query. This makes web searching an especially challenging and exciting area in the field of ▶ information retrieval.

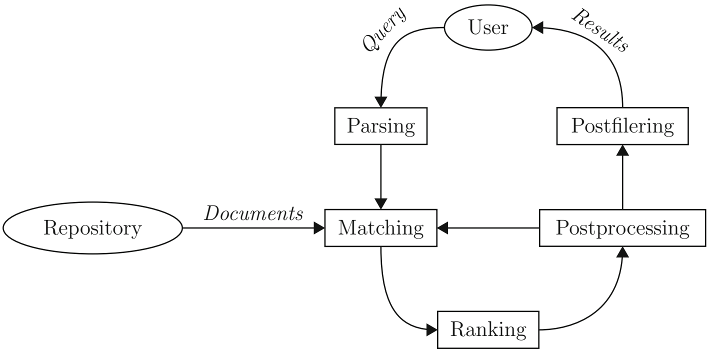
In Andrei (2002), it is suggested that web queries can be classified in three classes.

- **Navigational queries** expect the search to return a particular URL. For instance, http://www.cityrail.info is probably the expected result to the query `Cityrail` for a Sydneysider.
- **Transactional queries** expect the search to return links to sites that offer further interaction, for example, for online shopping or to download music. For instance, http://www.magickeys.com/books/, where books for young children are available for download, is probably a good result to the query `children stories`.
- **Informational queries** expect the search to reveal a piece of information that is the correct answer to a question. This piece of information can be immediately provided in the page where theresults of the search are displayed, as, for instance, `Bern` for the query `capital of switzerland`. Or it can be provided in the pages accessible from the first links returned by the search, as for instance, `Italy` that is easily found in the web page accessed from the first link returned in response to the query `football world champion 1982`.

Answering an informational query with the information itself, rather than with links to documents where the information is to be found, is one of the most difficult challenges that search engine developers have addressed. Some argue that the final goal is to deliver the best possible content that a user would like to have in a given moment and that instead of pulling information, information be pushed to the user depending on the context (Ricardo and Prabhakar 2010).

Structure of the Learning System

The general structure of a search engine can be illustrated as follows:



A ▶ string matching algorithm is applied to the parsed query issued by the user and to an indexed representation of a set of documents, resulting in a ranked subset of the latter. This ranked set of documents can be subjected to a postprocessing procedure whose aim is to improve the results by either refining the query or by analyzing further the documents, possibly over many iterations, until the results stabilize and can be returned to the user, following a postfiltering procedure to display the information appropriately.

Retrieval Methods

What distinguishes search engines from other information retrieval applications is the existence of hyperlinks between documents. All techniques developed in the field of information retrieval are potentially relevant for extracting information from the web, but will benefit from a proper analysis of the cross-reference structure. That is, to measure the degree of relevance of a document to a given query, one can take advantage of a prior ranking of all documents independent of that query or any other, following a sophisticated version of the PageRank (Lawrence et al. 1999) link analysis algorithm. One of the simplest versions of the algorithm recursively defines the PageRank PR(T) of a page T which pages $T_1, \ldots, T_n$ point to, among the $c_1, \ldots, c_n$ pages $T_1, \ldots, T_n$ point to, respectively, as

$$\frac{1-d}{N} + d(\mathrm{T}_1/c_1 + \cdots + \mathrm{T}_\mathrm{n}/c_n)$$

where N is the total number of pages and d, a *damping factor*, represents the probability that a user decides to follow a link rather than randomly visit another page; normalizing the solution so that the PageRanks of all pages add up to 1, PR(T) then represents the probability that a user visits T by clicking on a link.

Boolean retrieval is one of the simplest methods to retrieve a set of documents that match exactly a query expressed as a Boolean combination of keywords. The match is facilitated by using an *inverted file* indexing structure which associates every possible keyword with links to the documents in which it occurs (Justin and Alistair 2006). If extra information is kept on the occurrences of keywords in documents (number of occurrences, part of the document in which they occur, font size and font type used for their display, etc.), then the results can also be ranked. But *best match* models, as opposed to *exact match* models, are better suited to producing ranked results. The *vector space* model is one of the earliest and most studied models of this kind. It represents documents and queries as vectors over a space each of whose dimensions represents a possible keyword and measures the similarity between the vectors $\vec{q}$ and $\vec{d}$ that record for each keyword whether it occurs at least once in query and document, respectively, as the cosine of the angle formed by $\vec{q}$ and $\vec{d}$, namely,

$$\frac{\vec{q}.\vec{d}}{\|\vec{q}\|.\|\vec{d}\|},$$

that is all the most closer to 1 that query and document have more in common. The *term-frequency-inverse-document-frequency* (tf-idf) model refines the encoding given by $\vec{d}$ by replacing a value of 1 in the ith dimension, indicating the existence of an occurrence of the ith keyword in $\vec{d}$, with

$$c_1.\log\left(\frac{N}{c_2}\right)$$

where c_1 is the number of occurrences of the ith keyword in the document, N is the total number of documents, and c_2 is the number of documents in the whole collection that contains at least one occurrence of the ith keyword; so more weight is given to keywords that occur more and that occur "almost exclusively" in the document under consideration. One of the most obvious issues with this approach is that the number of dimensions is huge and the vectors are sparse. Another important issue is that set of vectors determined by the set of keywords is not orthogonal and not even linearly independent, because two given keywords can be synonyms (sick and ill), not semantically related (garlic and manifold), or more or less semantically related (wheel and tire).

The *extended vector space* model (Wong et al. 1987) addresses this issue assuming that the similarity between two keywords is captured by the symmetric difference between the set of documents that contain a keyword and the set of documents that contain the other, ranging from identical sets (similar keywords) to disjoint sets (unrelated keywords). Let $D_1, \ldots, D_{N'}$ be an enumeration of the quotient relation over the set of all documents such that two documents are equivalent if they contain precisely the same keywords (so N' is at most equal to N, the number of documents in the whole collection). Conceive an N'-dimensional vector space S which $D_1, \ldots, D_{N'}$ is a basis of. Associate the ith keyword with the vector $\vec{v_i}$ of S defined as $\frac{1}{\sqrt{w_1^2+\cdots+w_{N'}^2}}(w_1, \ldots, w_{N'})$ where for all nonzero

$k \leq N'$, w_k is the number of occurrences of the ith keyword in all documents that belong to class D_k. Then associate a document with the vector $\vec{d}$ of S defined as $\alpha_1 \vec{v}_1 + \cdots + \alpha_{N''} \vec{v}_{N''}$ where N'' is the number of keywords, and for all nonzero $k \leq N''$, α_k is the number of occurrences of the ith keyword in that document, and associate a query with the vector $\vec{q}$ of S defined as $\beta_1 \vec{v}_1 + \cdots + \beta_{N''} \vec{v}_{N''}$ where for all nonzero $k \leq N''$, β_k is equal to 1 if the ith keyword occurs in the query and to 0 otherwise. The similarity between $\vec{q}$ and $\vec{d}$ is then measured as described for the simple vector space method.

The *topic-based vector space* model (Jörg and Dominik 2003) also replaces the original vector space with a different vector space of a different dimension, addressing the issue of nonorthogonality between keywords, thanks to *fundamental topics*, assumed to be pairwise independent, using ontologies; the fundamental topics then provide the vector basis which a given keyword is a linear combination of. So the topic-based vector space model conceives the meaning of words as the semantic relationships that emerge from the common use of a language by the members of a given community, whereas the extended vector space model conceives the meaning of words as the syntactic relationship of term co-occurrence with respect to the repository of documents being processed.

Probabilistic retrieval frameworks aim at estimating the probability that a given document is relevant to a given query. Given a keyword w, denote by p_w^+ the probability that w occurs in a document relevant to w, and denote by p_w^- the probability that w occurs in a document not relevant to w. Many probabilistic retrieval frameworks then define the relevance of a document to a query as follows, where $w_1, \ldots, w_n$ are the keywords that occur both in the query and in the document:

$$\sum_{i=1}^{n} \log\left(\frac{p_{w_i}^+ (1 - p_{w_i}^-)}{p_{w_i}^- (1 - p_{w_i}^+)}\right).$$

This quantity increases all the more that the document contains more words more likely to occur in relevant documents and more words less likely to occur in irrelevant documents. Different frameworks suggest different ways to evaluate the values of $p_{w_i}^+$ and $p_{w_i}^-$. For instance, p_i is sometimes assumed to be constant and $p_{w_i}^-$ defined as n_i/N where N is the total number of documents and n_i the number of documents in which w_i occurs, capturing the fact that a document containing a keyword appearing in few other documents is likely to be relevant to that keyword, in which case the previous formula can be rewritten

$$c \times \sum_{i=1}^{n} \log\left(\frac{N - n_i}{n_i}\right).$$

for some constant c. More sophisticated methods have been developed to better estimate the probabilities, such as the *Okapi weighting document score* (Stephen et al. 1999) which defines the relevance of a document to a query as

$$\sum_{i=1}^{n} \log\left(\frac{N - n_i + 0.5}{n_i + 0.5}\right). \\ \frac{(k_1 + 1)c_i}{\left(k_1(1-b) + b\frac{l}{\beta}\right) + c_i} \cdot \frac{(k_3 + 1)d_i}{k_3 + d_i}$$

where the notation is as above, with the addition of c_i to denote the number of occurrences of w_i in the document, d_i to denote the number of occurrences of w_i in the query, l to denote the number of bytes in the document, β to denote the average number of bytes in a document, and b, k_1, and k_3 to denote constants.

Query Classification

The development of effective methods of information retrieval from web resources requires a good understanding of users' needs and practice. In Karen (2007a), the following questions are identified as being especially relevant toward gaining such an understanding:

> *What characterizes the queries that end users submit to online IR systems? What search features do people use? What features would enable them to improve on the retrievals they have*

> *in hand? What features are hardly ever used? What do end users do in response to the systems retrievals?*

This paper indicates that many of the basic features of information retrieval systems are poorly used. For instance, less than 15 %, 3 %, and 2 % of queries make use of the *and*, *or*, and *not* Boolean operators, respectively, and less than 15 % of queries of enclosing quotes; the wrong syntax is often used, resulting in incorrect use of advanced search features in one third of the cases; less than 10 % of queries take advantage of ▶ relevance feedback. Based on those findings, the second part (Karen 2007b) of the article suggests *two-dozen new research questions* for researchers in information retrieval while noting that about 70 % of users are satisfied with their search experience.

Evaluating search satisfaction has received lots of attention. In Steve et al. (2005), both explicit and implicit measures of satisfaction are collected. Explicit measures are obtained by prompting the user to evaluate a search result as satisfying, partially satisfying, or not satisfying and similarly to evaluate satisfaction gained from a whole search session. Implicit measures are obtained by recording mouse and keyboard actions, time spent on a page, scrolling actions and durations, number of visits to a page, position of page in results list, number of queries submitted, number of results visited, etc. A Bayesian model can be used to infer the relationships between explicit and implicit measures of satisfaction. This paper reports on two ▶ Bayesian networks that were built to predict satisfaction for individual page visits and satisfaction for entire search sessions—w.r.t. the feedback obtained from both kinds of prompts—with evidence that a combination of well-chosen implicit satisfaction measures can be a good predictor of explicit satisfaction. Referring to the categorization of web queries in Andrei (2002) as *user goals*, it is proposed in Uichin et al. (2005) to build *click distributions* by sorting results to a query following the numbers of clicks they received from all users and suggested that highly skewed distributions should correspond to navigational queries, while flat distributions should correspond to informational queries. The same kind of considerations are also applied to *anchor-link distributions*, the anchor-link distribution of a query being defined as the function that maps a URL to the number of times that URL is the destination of an anchor that has the same text as the query.

Finer techniques of query classification are proposed in Steven et al. (2007), where a rule-based automatic classifier is produced from *selectional preferences*. A query consisting of at least two keywords is split into a head x and a tail y and then converted into a *forward* pair (x, u) and a *backward* pair (u, y) where u represents a category, that is, a generic term that refers to a list of semantically related words in a thesaurus. For instance, the query "interest rate" can (only) be split into (interest, rate) and converted to the forward pair (interest, personal finance) where "personal finance" denotes the list consisting of the terms "banks," "rates," "savings," etc; so the first keyword—"interest"–provides context for the second one. Given a large query log, the *maximum likelihood estimate* (MLE) of $P(u/x)$, the probability that a query decomposed as (x, z) is such that z belongs to category u, is defined as the quotient between the number of queries in the log that have (x, u) as a forward pair and the number of queries in the log that can be decomposed as (x, z). This allows one to write a forward rule of the form "x Y classified as u with weight p" where p is the MLE of $P(u/x)$, provided that the *selectional preference strength* of x be above some given threshold. The rule can then be applied to incoming queries, such as "interest only loan" by matching a final or initial segment of the query—depending on whether forward or backward rules are under consideration—and suggest possible classifications; with the running example, "interest only loan" would then be classified as "personal finance with weight p" if a forward rule of the form "interest *Y* classified as personal finance with weight p" had been discovered. Such a classification can then be used to rewrite the query or to send it to an appropriate database backend if many domain-specific databases are available.

Cross-References

- ▶ Bayesian Methods
- ▶ Classification
- ▶ Covariance Matrix
- ▶ Rule Learning
- ▶ Text Mining

Recommended Reading

Baeza-Yates R, Raghavan P (2010) Next Generation Web Search. In: Ceri S, Brambilla M (eds) Next generation Web search, Springer Verlag, Berlin, Heidelberg, pp 11–23

Becker J, Kuropka D (2003) Topic-based vector space model. In: Abramowicz W, Klein G (eds) Proceedings of the 6th international conference on business information systems, Colorado Springs, pp 7–12

Beitzel SM, Jensen EC, Lewis DD, Chowdhury A, Frieder O (2007) Automatic classification of web queries using very large unlabeled query logs. ACM Trans Inf Syst 25(2) Article 9, pp 1–29

Broder A (2002) A taxonomy of web search. SIGIR Forum 36(2):3–10

Fox S, Karnawat K, Mydland M, Dumais S, White T (2005) Evaluating implicit measures to improve web search. ACM Trans Inf Syst 23(2):147–168

Lee U, Liu Z, Cho J (2005) Automatic identification of user goals in web search. In: WWW'05: proceedings of the 14th international conference on World Wide Web, Chiba, pp 391–400

Markev K (2007a) Twenty-five years of end-user searching, part 1: research findings. J Am Soc Inf Sci Technol 58(8):1071–1081

Markev K (2007b) Twenty-five years of end-user searching, part 2: future research directions. J Am Soc Inf Sci Technol 58(8):1123–1130

Page L, Brin S, Motwani R, Winograd T (1999) The pagerank citation ranking: bringing order to the web. Technical report, Stanford University

Robertson SE, Walker S, Beaulieu M (1999) Okapi at TREC–7: automatic ad hoc, filtering, VLC and filtering tracks. In: Voorhees E, Harman D (eds) Proceedings of the Seventh Text REtrieval Conference, pp 253—264

Wong SKM, Ziarko W, Raghavan VV, Wong PCN (1987) On modeling of information retrieval concepts in vector spaces. ACM Trans Database Syst 12(2):299–321

Zobel J, Moffat A (2006) Inverted files for text search engines. ACM Comput Surv 38(2)2:1–55

Selection of Algorithms, Ranking Learning Methods

▶ Metalearning

Self-Adaptive Systems

▶ Metalearning

Self-Organizing Feature Maps

▶ Self-Organizing Maps

Self-Organizing Maps

Samuel Kaski
Helsinki University of Technology, Helsinki, Finland

Synonyms

Kohonen maps; Self-organizing feature maps; SOM

Definition

Self-organizing map (SOM), or Kohonen Map, is a computational data analysis method which produces nonlinear mappings of data to lower dimensions. Alternatively, the SOM can be viewed as a ▶ clustering algorithm which produces a set of clusters organized on a regular grid. The roots of SOM are in neural computation (see ▶ neural networks); it has been used as an abstract model for the formation of ordered maps of brain functions, such as sensory feature maps. Several variants have been proposed, ranging from dynamic models to Bayesian variants. The SOM has been used widely as an engineering tool for data analysis, process monitoring, and information visualization, in numerous application areas.

Motivation and Background

The SOM (Kohonen 1982, 2001) was originally introduced in the context of modeling of how the spatial organization of brain functions forms. Formation of feature detectors selective to certain sensory inputs, such as orientation-selective visual neurons, had earlier been modeled by ▶ competitive learning in neural networks, and some models of how the feature detectors become spatially ordered had been published (von der Malsburg 1973). The SOM introduced an *adaptation kernel* or *neighborhood function* that governs the adaptation in such networks; while in plain competitive learning only the winning neuron that best matches the inputs adapts, in SOM all neurons within a local neighborhood of the winner learn. The neighborhood is determined by the neighborhood function. The SOM is an algorithm for computing such ordered mappings.

While some of the motivation of the SOM comes from neural computation, its main uses have been as a practical data analysis method. The SOM can be viewed as a topographic vector quantizer, a nonlinear projection method, or a clustering method. In particular, it is a clustering-type algorithm that orders the clusters. Alternatively, it is a nonlinear projection-type algorithm that clusters, or more specifically quantizes, the data.

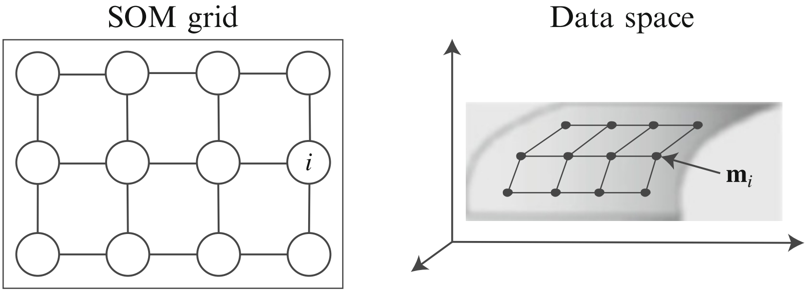
The SOM was very popular in the 1990s and still is; it is intuitively relatively easily understandable, yet hard to analyze thoroughly. It connects many research traditions and works well in practice. An impressive set of variants have been published over the years, of which probabilistic variants (e.g., Bishop et al. (1998) and Heskes (2001)) are perhaps closest to the current mainstream machine learning. While there currently are excellent alternative choices for many of the specific tasks SOMs have been applied for over the years, even the basic SOM algorithm is still viable as a versatile engineering tool in data-analysis tasks.

Structure of Learning System

The SOM consists of a regular grid of nodes (Fig. 1). A *model* of data has been attached to each node. For vector-valued data $\mathbf{x} = [x_1, \ldots, x_d]^T$, the models are vectors in the same space; the model at the ith node is $\mathbf{m}_i = [m_{i1}, \ldots, m_{id}]$. The models define a mapping from the grid to the data space. The coordinates on the grid are uniquely determined by the index i of a node, and the model $\mathbf{m}_i$ gives the location in the data space. The whole grid becomes mapped into an “elastic net” in the data space. While being a mapping from the grid to the input space, the SOM defines a projection from the input space to the discrete grid locations as well; each data point is projected to the node having the closest model.

The original online SOM algorithm updates the model vectors toward the current input vector at time t,

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{ci}(t)(x(t) - \mathbf{m}_i(t)).$$



Self-Organizing Maps, Fig. 1 A schematic diagram showing how the SOM grid of units (*circles* on the *left*, neighbors connected with *lines*) corresponds to an “elastic net” in the data space. The mapping from the grid locations, determined by the indices i, to the data space is given by the model vectors $\mathbf{m}_i$ attached to the units i

Here c is the index of the unit having the closest model vector to $\mathbf{x}(t)$, and $h_{ci}(t)$ is the neighborhood function or adaptation kernel. The kernel is a decreasing function of the distance between the units i and c on the grid; it forces neighboring units to adapt toward similar input samples. The height and width of h are decreasing functions of time t. In an iteration over time and over the different inputs, the model vectors become ordered and specialize to represent different regions of the input space.

The online version of ▶ K-means clustering is a special case of the SOM learning rule, where only the closest model vector is adapted. That is, the neighborhood function is $h_{ci}(t) = \alpha(t)$ for $i = c$ and $h_{ci} = 0$ otherwise. Here $\alpha(t)$ is the adaptation coefficient, a decreasing scalar. In short, K-means and SOM use the prototypes in the same way, but in SOM the prototypes have an inherent order that stems from fixing them onto a grid and updating the prototypes to represent both the data mapped to themselves and to their neighbors.

A neural interpretation of the SOM adaptation process is that the nodes are feature detector neurons or processing modules that in a ▶ competitive learning process become specialized to represent different kinds of inputs. The neighborhood function is a plasticity kernel that forces neighboring neurons to adapt at the same time. The kernel transforms the discrete set of feature detectors into feature maps analogous to ordered brain maps of sensor inputs, and more generally to maps of more abstract properties of the input data.

A third interpretation of the SOM is as a vector quantizer. The task of a vector quantizer is to encode inputs with indexes of prototypes, often called codebook vectors, such that a distortion measure is minimized. If there is noise that may change the indexes, the distribution of the noise should be used as the neighborhood function, and then the distortion becomes minimized by a variant of SOM (Luttrell 1994). In summary, the SOM can be viewed as an algorithm for producing codebooks ordered on a grid.

While it has turned out to be hard to rigorously analyze the properties of the SOM algorithm (Fort 2006), its fixed points may be informative. In a fixed point the models must fulfill

$$m_i = \frac{\sum_x h_{c(x),i}\,\mathbf{x}}{\sum_{\mathbf{x}} h_{c(\mathbf{x},i)}},$$

that is, each model vector is in the centroid of data projected to it and its neighbors. The definition of a *principal curve* (Hastie et al. 2001), a nonlinear generalization of principal components (see ▶ Principal Component Analysis), essentially is that the curve goes through the centroid of data projected to it. Hence, one interpretation of the SOM is a discretized, smoothed, nonlinear generalization of principal components. In short, SOMs aim to describe the variation in the data nonlinearly with their discrete grids.

Finally, a popular prototype-based classifier, ▶ learning vector quantization (LVQ) (Kohonen 2001), can be loosely interpreted as a variant of SOMs, although it does not have the neighborhood function and hence, the prototypes do not have an order.

Programs and Data

The SOM has been implemented in several commercial packages and as freeware. Two examples, SOM_PAK written in C and Matlab SOM Toolbox (http://www.cis.hut.fi/research/software) came from Kohonen's group.

Applications

The SOM can be used as a nonlinear dimensionality reduction method, by projecting each data vector into the grid location having the closest model vector. An image of the grid can be used for *information visualization*. Since all grid locations are clusters, the SOM display actually visualizes an ordered set of clusters, or a quantized image of the principal manifold in data. More specifically, the SOM units can be thought of as subclusters, and data clusters may form larger areas on the SOM grid.

SOM-based visualizations can be used for illustrating the proximity relationships of data vectors, such as documents in the WEBSOM document maps (Kohonen et al. 2000), or monitoring the change of a system such as an industrial process or the utterances of a speaker, as a trajectory on the SOM display. More applications can be found in a collected bibliography (the latest one is Pöllä et al. 2009).

Cross-References

- ▶ ART
- ▶ Competitive Learning
- ▶ Dimensionality Reduction
- ▶ Hebbian Learning
- ▶ K-means Clustering
- ▶ Learning Vector Quantization

Recommended Reading

Bishop CM, Svensén M, Williams CKI (1998) GTM: the generative topographic mapping. Neural Comput 10:215–234

Fort JC (2006) SOM's mathematics. Neural Netw 19: 812–816

Hastie T, Tibshirani R, Friedman J (2001) The elements of statistical learning. Springer, New York

Heskes T (2001) Self-organizing maps, vector quantization, and mixture modeling. IEEE Trans Neural Netw 12:1299–1305

Kohonen T (1982) Self-organized formation of topologically correct feature maps. Biol Cybern 43:59–69

Kohonen T (2001) Self-organizing maps, 3rd edn. Springer, Berlin

Kohonen T, Kaski S, Lagus K, Salojärvi J, Honkela J, Paatero V et al (2000) Self organization of a massive document collection. IEEE Trans Neural Netw 11: 574–585

Luttrell SP (1994) A Bayesian analysis of self-organizing maps. Neural Comput 6:767–794

Pöllä M, Honkela T, Kohonen T (2009) Bibliography of self-organizing map (SOM) papers: 2002–2005 addendum. Report TKK-ICS-R23, Helsinki University of Technology, Department of Information and Computer Science, Espoo

von der Malsburg C (1973) Self-organization of orientation sensitive cells in the striate cortex. Kybernetik 14:85–100

Semantic Annotation of Text Using Open Semantic Resources

Stefano Pacifico[1], Janez Starc[1], Janez Brank[1], Luka Bradesko[1], and Marko Grobelnik[2]
[1]Jožef Stefan Insitute, Ljubljana, Slovenia
[2]Artificial Intelligence Laboratory, Jožef Stefan Insitute, Ljubljana, Slovenia

Abstract

In this article, we present the topic of semantic annotation of text using open semantic resources. We present an introduction to the concept and its history, provide basic notions around semantic annotations and open semantic resources, in particular illustrating commonly used open semantic resources repositories such as Wikipedia, Wordnet, or DB-Pedia. Further, we discuss the issues around creating open semantic resources, both from the annotation perspective, and from the format perspective. Finally, we introduce two well-known semantic annotation tasks, entity linking (or named entity disambiguation), and semantic parsing, with corresponding sample implementations, explaining in particular how they work and how make use of open semantic resources.

Synonyms

Synopsis: Text annotation is the association of metadata to fragments of text. When the associated metadata provides a model for interpreting the fragment of text, we talk about *semantic annotation of text*. A common case is the use of knowledge extracted from thesauri, or ontologies to provide the machine with structure and inference mechanisms for the meaning of the text under consideration (Völkel et al. 2006).

The spreading of the Semantic Web (Berners-Lee et al. 2001) and Linked Open Data (http://linkeddata.org/) movements led to the adoption of open data sets having well defined for semantic annotations. Examples include disambiguation of word senses using knowledge bases or the analysis of the actions and roles described in a fragment of text.

Introduction

Semantic annotations are used to improve the quality of several applications including text classification, question answering, machine reading, and understanding. They are used in a variety of domains including, media, genomics marketing, and social policies. (Uren et al. 2006).

Using *open semantic resources* for semantic annotation has become common in the state-of-the-art semantic annotation systems. Open semantic resources provide an accessible framework for people and applications to link structured semantic information to text, effectively allowing ideas like the Semantic Web to come alive, offering an exponentially large set of annotated documents (Völkel et al. 2006). An exemplar is Wikipedia, which provides a large corpus of semi-structured text associated with concepts and entities, wide multilingual coverage, and a large network of internal links providing both structure and a dictionary of surface forms for the linked concepts and entities. At the same time, Wikipedia offers limited breadth in specific domains and an uneven distribution of quality and thoroughness throughout concepts and languages (Bradesko et al. 2015).

Background Knowledge

In this section, we present examples of different types of open semantic resources. Further, we show some of the problems typically encountered when obtaining annotated corpora and finally problems associated with the representation of annotations.

The most used repositories of open semantic resources fall in two categories, *lexical databases* and *knowledge bases*. Lexical databases are repositories of words and metadata associated with them. Knowledge bases, instead, are organized collections of structured data and their relationships. Examples of lexical databases include *WordNet* and *FrameNet* (https://framenet.icsi.berkeley.edu/fndrupal/about).

WordNet is a lexical database where words are related through synonymy and are grouped into *synsets*, sets of terms having the same meaning. Synsets are related by hyperonimy (ISA relationship), meronimy (part-whole relationship), and antinomy (opposite relationship), among the others (Fellbaum 2005).

FrameNet is a lexical database that is both human and machine readable, containing more than 10,000 word senses, and more than 170,000 manually annotated sentences provide a unique training data set for *semantic role labeling* (Gildea and Jurafsky 2002). FrameNet is based on the frame semantics theory (Baker et al. 1998) in which meaning is conveyed with a semantic frame structure that includes the type of an event, the participants, and their roles and relations. For example, the frame *apply heat* is used in the context of cooking and contains *frame elements* (FEs): *cook*, *food*, *heating instrument*, and *container*. Words such as *fry*, *bake*, and *boil*, are called *lexical units* (LUs) of the *apply heat* frame and are used to detect if the sentence should be interpreted in the context of the specified frame.

Knowledge bases, instead, constitute part of the critical infrastructure of a knowledge system. Knowledge bases are traditionally studied in knowledge representation and reasoning that, according to Brachman *et al.*, is "the area of artificial Intelligence (AI) concerned with how knowledge can be represented symbolically and manipulated in an automated way by reasoning programs" (Brachman and Levesque 2004). Notable open knowledge bases used in semantic annotation systems include DBPedia, YAGO, and Wikidata. DBPedia is a large-scale, multilingual knowledge base created by extracting structured data from Wikipedia editions in 111 languages. The DBPedia project maps Wikipedia infoboxes from 27 different language editions to a single shared ontology consisting of 320 classes and 1,650 properties. DBPedia has more than 27 million links to other open data repositories via the `owl::sameas` relation, including common sense ontologies and government data (Lehmann et al. 2015). YAGO is another ontology automatically built from Wikipedia, with links to Geonames (http://geonames.org) and WordNet. Among other things, it features spatial and temporal knowledge and inference, and as in the case of DBpedia, it covers different languages (Mahdisoltani et al. 2015).

The other knowledge base mentioned, Wikidata, was created as the knowledge base of Wikipedia. Later, it absorbed the popular collaborative knowledge base Freebase (Bollacker et al. 2008). Wikidata is the central data management platform of Wikipedia. The data are highly interlinked and connected to many other data sets, and, since 2014, RDF exports that connect Wikidata to the Linked Data Web are available (Erxleben et al. 2014). Other open knowledge bases worth mentioning include BabelNet (http://babelnet.org) and OpenCyc (http://opencyc.org).

While knowledge bases provide the resources for systems to perform semantic annotations, corpora annotated with those resources are necessary to train evaluate the performance of systems. Manually annotated corpora are difficult and expensive to build, as exemplified by the CLEF corpus (Roberts et al. 2007), a set of structured and unstructured annotated health records. Examples of corpora annotations with open semantic resources include the data sets released for the shared tasks of the Conference on Natural Language Learning (CoNLL) (Surdeanu et al. 2008; Hajič et al. 2009) as the treebanks made available for semantic parsing or named entities and semantic role labeling. As mentioned before, Wikipedia, together with its derivative knowledge bases, constitutes one of the largest manually annotated corpora, albeit the ever-evolving and with varying degrees of quality and consistency in the annotations.

Finally, we want to touch on the choice of the representation for semantic annotations. First, we should distinguish between format (i.e., *how* to represent, e.g., XML) and model (i.e., *what* to represent, e.g., RDF). Annotations formats and models vary by syntax style (e.g., markup vs declarative language), expressive power, or computational complexity. Notably, RDF, the Resource Description Framework (http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/), and RDF Schema, a data-modeling oriented extension of RDF, provide the mechanics of the popular triple representation: subject, predicate, and object. After the introduction of RDF, other logical representations such as OWL (W3C 2009), Description Logics (Baader et al. 2010) and Schema.org (http://schema.org/docs/about.html) followed, with the aim of implementing the Semantic Web vision. Formalisms, like the Knowledge Interchange Format (http://www.ksl.stanford.edu/knowledge-sharing/kif/), propose higher-level constructs at the price of a more complex and verbose language, while others, like the Knowledge Annotation Format (Bosma et al. 2009), aim to address nesting and integration of different annotation sources.

Structure of Learning Systems

Common types of semantic annotations include, but are not limited to, the *disambiguation* of terms (words or phrases) and to the *parsing* of sentences from a semantic point of view. We will focus on these two in the remainder of this section.

The task of disambiguating the meaning of fragments of text can range from generic word-sense disambiguation (Navigli 2009), in which the system is given the task of assigning a unique sense to a word or expression from a specific set of canonical senses to more specialized problems, such as the one commonly known as *wikification*, the task of linking words or expression to a Wikipedia entry (Mihalcea and Csomai 2007; Cucerzan 2007; Milne and Witten 2008), or named-entity disambiguation, that is, word-sense disambiguation applied to entities bearing a proper name (e.g., people, organizations, or locations) (Hoffart et al. 2011; Nguyen et al. 2014; Mendes et al. 2011).

Examples of features used by algorithms for disambiguation problems include *prior probability*, the probability that a given surface form refers to a specific sense or entity, calculated from annotated training data sets; *similarity*, in which segment of texts are compared with gloss overlap or other similarity measures (Lesk 1986; Banerjee and Pedersen 2002); *co-occurrence*, the learned probability that a pair of senses or entities appear together in a document; and *coherence* a calculated measure of relatedness between senses or entities (e.g., number of incoming links shared by two Wikipedia entries or gloss overlap of the description of two entities). These features can be directly used in classification algorithms like support vector machines or combined to create different features. For examples, relatedness graphs between mentions and senses can be created, using features like the ones mentioned, to calculate the weights of the graph. In that spirit, recent state-of-the-art solutions apply algorithms similar to *page rank* to such graphs. As a result, the annotations with the highest ranking form a reasonably coherent set of annotations (Pershina et al. 2015). Other unsupervised approaches include identifying senses of n-grams based on contextual windows of text around the candidate n-gram (Navigli 2009).

Disambiguation algorithms typically follow three stages: (1) finding the fragments of text that require disambiguation, (2) producing a list of candidate senses, and (3) choosing the target sense for the disambiguation. Implementations vary in how they tackle these steps. For example, named-entity recognition algorithms can be run on the text to identify the surface forms of named entities requiring disambiguation, as in Hoffart et al. (2011), while wikification algorithms rely on either user selection or by automatically selecting n-grams that appear as anchor links in Wikipedia with a probability higher than a given threshold (Cucerzan 2007; Milne and Witten 2008). Candidates are selected by reverse lookup in dictionaries built from annotated examples or using lexical similarity functions such locality-sensitive hashing to address wrong or alternative spellings (e.g., *traveling* vs *travelling*).

The other common semantic annotation task is semantic parsing. Semantic parsing is the process of mapping fragments of text into a representation that reflects their meaning. It can also be seen as the reverse of language generation. Representation languages range from more expressive, such as first-order logic and lambda calculus, to more simple database query languages designed specifically for a small domain, like Geoquery (Zelle and Mooney 1996).

Parsing is usually divided into assigning meaning representations to lexical units and composing these representations into a single one. In some cases, syntactic parsing is performed on the input text, and the results are used to derive the semantics from the syntax tree (Poon and Domingos 2009). Other methods perform syntactic and semantic parsing together using formalisms like combinatory categorial grammars (CCG) (Zettlemoyer and Collins 2009) or different variations of context-free grammars (CFG)(Wong and Mooney 2006).

Early semantic parsers were mostly handcrafted (Warren 1981). It turned out that developing training corpora is not more difficult than manually designing robust semantic parsers. With the availability of training corpora and improved hardware, learning approaches started to emerge. These approaches automatically learn from sets of sentence meaning representation pairs. Usually, an expectation-maximization-like algorithm is used to train a model, which selects the most likely representation for a given sentence.

Training corpora are usually expensive to develop and limited to a particular domain. This led to the development of methods that are large scale, domain independent, and to the use of other forms of supervision. The level of supervision ranges from unsupervised methods (Poon and Domingos 2009), which bootstrap from a small set of training pairs, to weakly supervised methods (Cai and Yates 2013), which use knowledge from a target knowledge base, like Freebase, to methods that learn from question answer pairs (Berant et al. 2013). Lately, methods that use neural models to model the semantics have been developed (Bordes et al. 2012). The evaluation of semantic parsing can be performed by manually inspecting the generated meaning representations or in a question answer setting, where questions are mapped to database queries. The queries are then executed to obtain a set of answers, which are compared to the golden set of answers.

Notable systems for semantic annotations include AIDA, a system developed by the Max Planck Institute for named-entity disambiguation, able to link named entities contained in text to entities in the YAGO2 ontology; DBPedia Spotlight (https://github.com/dbpedia-spotlight/dbpedia-spotlight/wiki), a system from Free University of Berlin for extracting entities from text and linking them to the DBpedia ontology; and Babelfy (http://babelfy.org).

Recommended Reading

- Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen F"urstenau Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. Robust Disambiguation of Named Entities in Text. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, EMNLP 2011, Edinburgh, Scotland, pages 782–792, 2011.
- David Milne and Ian H. Witten. Learning to link with wikipedia. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 509–518. ACM, 2008.
- Roberto Navigli. Word sense disambiguation: A survey. ACM Comput. Surv., 41(2):10:1–10:69, February 2009.
- Luke S Zettlemoyer and Michael Collins. Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*: Volume 2, pages 976–984. Association for Computational Linguistics, 2009.

Baader F, Calvanese D, McGuiness DL, Nardi D, Patel-Schneieder PF (2010) The description logic handbook: theory, implementation, and applications. Cambridge University Press, Cambridge

Baker CF, Fillmore CJ, Lowe JB (1998) The Berkeley framenet project. In: Proceedings of the 17th international conference on computational linguistics. Association for computational linguistics, Montreal, vol 1, pp 86–90

Banerjee S, Pedersen T (2002) An adapted lesk algorithm for word sense disambiguation using wordnet. In: Proceedings of computational linguistics and intelligent text processing,third international conference, CICLing 2002, Mexico City, pp 136–145

Berant J, Chou A, Frostig R, Liang P (2013) Semantic parsing on freebase from question-answer pairs. In: EMNLP, Seattle, pp 1533–1544

Berners-Lee T, Hendler J, Lassila O (2001) The semantic web. Sci Am 284:34–43

Bollacker K, Evans C, Paritosh P, Sturge T, Taylor J (2008) Freebase: a collaboratively created graph database for structuring human knowledge. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data, SIGMOD'08. ACM, New York, pp 1247–1250

Bordes A, Glorot X, Weston J, Bengio Y (2012) Joint learning of words and meaning representations for open-text semantic parsing. In: International conference on artificial intelligence and statistics, La Palma, pp 127–135

Bosma W, Vossen P, Soroa A, Rigau G, Tesconi M, Marchetti A, Monachini M, Aliprandi C (2009) KAF: a generic semantic annotation format. In: Proceedings of the GL2009 workshop on semantic annotation, Pisa

Brachman RJ, Levesque HJ (2004) Knowledge representation and reasoning. The Morgan Kaufmann series in artificial intelligence series. Morgan Kaufmann, Burlington

Bradesko L, Starc J, Pacifico S (2015) Isaac bloomberg meets Michael bloomberg: better entity disambiguation for the news. In: Proceedings of the 24th international conference on World Wide Web companion WWW, companion volume, Florence, pp 631–635

Cai Q, Yates A (2013) Large-scale semantic parsing via schema matching and lexicon extension. In: ACL (1), Sofia. Citeseer, pp 423–433

Cucerzan S (2007) Large-scale named entity disambiguation based on wikipedia data. In: Proceeding of the 2007 joint conference on EMNLP and CNLL, Prague, pp 708–716

Erxleben F, Günther M, Krötzsch M, Mendez J, Vrandecic J (2014) Introducing Wikidata to the linked data web. In: The semantic web – ISWC 2014 – Proceedings of 13th international semantic web conference part I, Riva del Garda, pp 50–65

Fellbaum C (2005) Wordnet and wordnets. In: Brown K (ed) Encyclopedia of language and linguistics. Oxford, Elsevier, pp 665–670

Gildea D, Jurafsky D (2002) Automatic labeling of semantic roles. Comput Linguist 28:245–288

Hajič J, Ciaramita M, Johansson R, Kawahara D, Martí M, Màrquez L, Meyers A, Nivre J, Padó S, Štěpánek J et al (2009) The conll-2009 shared task: syntactic and semantic dependencies in multiple languages. In: Proceedings of the thirteenth conference on computational natural language learning: shared task. Association for computational linguistics, Stroudsburg, pp 1–18

Hoffart J, Yosef MA, Bordino I, Fürstenau H, Pinkal M, Spaniol M, Taneva B, Thater S, Weikum G

(2011) Robust disambiguation of named entities in text. In: Proceedings of the 2011 conference on empirical methods in natural language processing, EMNLP 2011, Edinburgh, pp 782–792

Lehmann J, Isele R, Jakob M, Jentzsch A, Kontokostas D, Mendes PN, Hellmann S, Morsey M, van Kleef P, Auer S, Bizer C (2015) DBpedia – a large-scale, multilingual knowledge base extracted from wikipedia. Semant Web J 6(2):167–195

Lesk M (1986) Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In: ACM special interest group for design of communication, Chicago, pp 24–26

Mahdisoltani F, Biega J, Suchanek F (2015) YAGO3: a knowledge base from multilingual Wikipedias. In: Proceeding of 7th biennial conference on innovative data systems research (CIDR 2015), Asilomar

Mendes PN, Jakob M, Garcia-Silva A, Bizer C (2011) Dbpedia spotlight: shedding light on the web of documents. In: Proceedings of the 7th international conference on semantic systems (I-semantics), Graz

Mihalcea R, Csomai A (2007) Wikify!: linking documents to encyclopedic knowledge. In: Proceedings of the sixteenth ACM conference on information and knowledge management, CIKM'07. ACM, New York, pp 233–242

Milne D, Witten IH (2008) Learning to link with wikipedia. In: Proceedings of the 17th ACM conference on information and knowledge management. ACM, Napa Valley, pp 509–518

Navigli R (2009) Word sense disambiguation: a survey. ACM Comput Surv 41(2):10:1–10:69

Nguyen DB, Hoffart J, Theobald M, Weikum G (2014) AIDA-light: high-throughput named-entity disambiguation. In: Linked data on the web at WWW2014, Seoul

W3C OWL Working Group (2009) OWL 2 web ontology language: document overview. W3C Recommendation, 27 October 2009. Available at http://www.w3.org/TR/owl2-overview/

Pershina M, He Y, Grishman R (2015) Personalized page rank for named entity disambiguation. In: Proceedings of the 2015 conference of the North American chapter of the association for computational linguistics: human language technologies. Association for computational linguistics, Denver, pp 238–243

Poon H, Domingos P (2009) Unsupervised semantic parsing. In: Proceedings of the 2009 conference on empirical methods in natural language processing, vol 1. Association for computational linguistics, Singapore, pp 1–10

Roberts A, Gaizauskas R, Hepple M, Davis N, Demetriou G, Guo Y, Kola JS, Roberts I, Setzer A, Tapuria A et al (2007) The clef corpus: semantic annotation of clinical text. In: AMIA annual symposium proceedings. American Medical Informatics Association, Chicago, vol 2007, p 625

Surdeanu M, Johansson R, Meyers A, Màrquez L, Nivre J (2008) The conll-2008 shared task on joint parsing of syntactic and semantic dependencies. In: Proceedings of the twelfth conference on computational natural language learning. Association for computational linguistics, Manchester, pp 159–177

Uren V, Cimiano P, Iria J, Handschuh S, Vargas-Vera M, Motta E, Ciravegna F (2006) Semantic annotation for knowledge management: requirements and a survey of the state of the art. Web Semant sci serv Agents World Wide Web 4(1):14–28

Völkel M, Krötzsch M, Vrandečić D, Haller H, Studer R (2006) Semantic wikipedia. In: Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh

Warren DHD (1981) Efficient processing of interactive relational data base queries expressed in logic. In: Proceedings of the seventh international conference on very large data bases, vol 7. VLDB Endowment, Cannes, pp 272–281

Wong YW, Mooney RJ (2006) Learning for semantic parsing with statistical machine translation. In: Proceedings of the main conference on human language technology conference of the North American chapter of the association of computational linguistics. Association for computational linguistics, New York, pp 439–446

Zelle JM, Mooney RJ (1996) Learning to parse database queries using inductive logic programming. In: Proceedings of the national conference on artificial intelligence, Portland, pp 1050–1055

Zettlemoyer LS, Collins M (2009) Learning context-dependent mappings from sentences to logical form. In: Proceedings of the joint conference of the 47th annual meeting of the ACL and the 4th international joint conference on natural language processing of the AFNLP, vol 2. Association for Computational Linguistics, Stroudsburg, pp 976–984

Semantic Mapping

► Text Visualization

Semi-naive Bayesian Learning

Fei Zheng[1] and Geoffrey I. Webb[2]
[1]Faculty of Information Technology, Monash University, Clayton, Melbourne, VIC, Australia
[2]Faculty of Information Technology, Monash University, Victoria, Australia

Definition

Semi-naive Bayesian learning refers to a field of ► Supervised Classification that seeks to enhance the classification and conditional probability estimation accuracy of ▶ naive Bayes by relaxing its attribute independence assumption.

Motivation and Background

The assumption underlying ▶ naive Bayes is that attributes are independent of each other, given the class. This is an unrealistic assumption for many applications. Violations of this assumption can render naive Bayes' classification suboptimal. There have been many attempts to improve the classification accuracy and probability estimation of naive Bayes by relaxing the attribute independence assumption while at the same time retaining much of its simplicity and efficiency.

Taxonomy of Semi-naive Bayesian Techniques

Semi-naive Bayesian methods can be roughly subdivided into five high-level strategies for relaxing the independence assumption.

- The first strategy forms an attribute subset by deleting attributes to remove harmful interdependencies and applies conventional naive Bayes to this attribute subset.
- The second strategy modifies naive Bayes by adding explicit interdependencies between attributes.
- The third strategy accommodates violations of the attribute independence assumption by applying naive Bayes to a subset of training set. Note that the second and third strategies are not mutually exclusive.
- The fourth strategy performs adjustments to the output of naive Bayes without altering its direct operation.
- The fifth strategy introduces hidden variables to naive Bayes.

Methods that Apply Naive Bayes to a Subset of Attributes

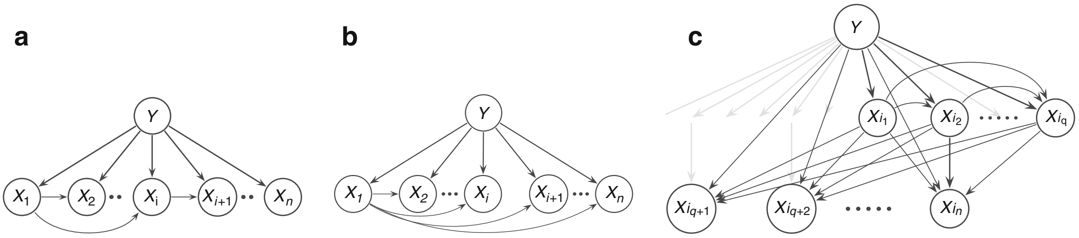
Due to the attribute independence assumption, the accuracy of naive Bayes is often degraded by the presence of strongly correlated attributes. Irrelevant attributes may also degrade the accuracy of naive Bayes, in effect increasing variance without decreasing bias. Hence, it is useful to remove both strongly correlated and irrelevant attributes.

Backward sequential elimination (Kittler 1986) is an effective wrapper technique to select an attribute subset and has been profitably applied to naive Bayes. It begins with the complete attribute set and iteratively removes successive attributes. On each iteration, naive Bayes is applied to every subset of attributes that can be formed by removing one further attribute. The attribute whose deletion most improves training set accuracy is then removed, and the process repeated. It terminates the process when subsequent attribute deletion does not improve training set accuracy. Conventional naive Bayes is then applied to the resulting attribute subset.

One extreme type of interdependencies between attributes results in a value of one being a generalization of a value of the other. For example, *Gender* = *female* is a generalization of *Pregnant* = *yes*. Subsumption resolution (SR) (Zheng et al. 2012) identifies at classification time pairs of attribute values such that one appears to subsume (be a generalization of) the other and delete the generalization. It uses the criterion $|T_{xi}| = |T_{x_i,x_j}| \geq u$ to infer that attribute value x_j is a generalization of attribute value x_i, where $|T_{x_i}|$ is the number of training cases with value x_i, $|T_{x_i}, x_j|$ is the number of training cases with both values, and u is a user-specified minimum frequency. When SR is applied to naive Bayes, the resulting classifier acts as naive Bayes except that it deletes generalization attribute-values at classification time if a specialization is detected.

Methods that Alter Naive Bayes by Allowing Interdependencies Between Attributes

Interdependencies between attributes can be addressed directly by allowing an attribute to depend on other non-class attributes. Sahami (1996) introduces the terminology of the z-dependence



Semi-naive Bayesian Learning, Fig. 1 Bayesian Network. (**a**) one-dependence classifier, (**b**) SuperParent one-dependence classifier and (**c**) z-dependence classifier $(z \geq 0)$

Bayesian classifier, in which each attribute depends upon the class and at most z other attributes. Figure 1 depicts methods in this group from the ▶ Bayesian Network perspective.

In Fig. 1a, each attribute depends on the class and at most one another attribute. ▶ Tree Augmented Naive Bayes (TAN) (Friedman et al. 1997) is a representative one-dependence classifier. It efficiently finds a directed spanning tree by maximizing the log-likelihood and employs this tree to perform classification. SuperParent TAN (Keogh and Pazzani 1999) is an effective variant of TAN.

A SuperParent one-dependence classifier (Fig. 1b) is a special case of one-dependence classifiers, in which an attribute called the SuperParent (X_1 in this graph), is selected as the parent of all the other attributes. ▶ Averaged One-Dependence Estimators (AODE) (Webb et al. 2005) selects a restricted class of one-dependence classifiers and aggregates the predictions of all qualified classifiers within this class. Maximum a posteriori linear mixture of generative distributions (MAPLMG) (Cerquides and Mántaras 2005) extends AODE by assigning a weight to each one-dependence classifier.

Two z-dependence classifiers ($z \geq 0$) are NBTree (Kohavi 1996) and lazy Bayesian rules (LBR) (Zheng and Webb 2000), both of which may add any number of non-class-parents for an attribute. In Fig. 1c, attributes in $\{X_{i_{q+1}}, \ldots, X_{i_n}\}$ depend on all the attributes in $\{X_{i_1}, \ldots, X_{i_q}\}$. The main difference between these two methods is that NBTree builds a single tree for all training instances while LBR generates a Bayesian rule for each test instance.

Methods that Apply Naive Bayes to a Subset of the Training Set

Another effective approach to accommodating violations of the conditional independence assumption is to apply naive Bayes to a subset of the training set, as it is possible that the assumption, although violated in the whole training set, may hold or approximately hold in a subset of the training set. NBTree and LBR use a local naive Bayes to classify an instance and can also be classified into this group. Locally weighted naive Bayes (LWNB) (Frank et al. 2003) applies naive Bayes to a neighborhood of the test instance, in which each instance is assigned a weight decreasing linearly with the Euclidean distance to the test instance. The number of instances in the subset is determined by a user-specified parameter. Only those instances whose weights are greater than zero are used for classification.

Methods that Calibrate Naive Bayes' Probability Estimates

Methods in this group make adjustments to the distortion in estimated posterior probabilities resulting from violations of independence assumption. Isotonic regression (IR) (Zadrozny and Elkan 2002) is a nonparametric calibration method which produces a monotonically increasing transformation of the probability outcome of naive Bayes. It uses a pair-adjacent violators algorithm (Ayer et al. 1955) to perform calibration. To classify a test instance, IR first finds the interval in which the estimated posterior probability fits and predicts the isotonic regression estimate of this interval as the calibrated posterior probability. Adjusted probability naive Bayesian classification (Webb and Pazzani 1998) makes adjustments to class probabilities, using a simple hill-climbing search to find adjustments that maximize the ▸ leave-one-out cross validation accuracy estimate. Starting with the conditional attribute-value frequency table generated by naive Bayes, iterative Bayes (Gama 2003) iteratively updates the frequency table by cycling through all training instances.

Methods that Introduce Hidden Variables to Naive Bayes

Creating hidden variables or joining attributes is another effective approach to relaxing the attribute independence assumption. Backward sequential elimination and joining (BSEJ) (Pazzani 1996) extends BSE by creating new Cartesian product attributes. It considers joining each pair of attributes and creates new Cartesian product attributes if the action improves leave-one-out cross validation accuracy. It deletes original attributes and also new Cartesian product attributes during a hill-climbing search. This process of joining or deleting is repeated until there is no further accuracy improvement. Hierarchical naive Bayes (Zhang et al. 2004) uses conditional mutual information as a criterion to create a hidden variable whose value set is initialized to the Cartesian product over all the value sets of its children. Values of a hidden variable are then collapsed by maximizing conditional log-likelihood via the ▸ minimum description length principle (Rissanen 1978).

Selection Between Semi-naive Bayesian Methods

No algorithm is universally optimal in terms of generalization accuracy. General recommendations for selection between semi-naive Bayesian methods is provided based on ▸ bias-variance tradeoff together with characteristics of the application to which they are applied.

Error can be decomposed into bias and variance (see ▸ bias variance decomposition). Bias measures how closely a learner is able to approximate the decision surfaces for a domain and variance measures the sensitivity of a learner to random variations in the training data. Unfortunately, we cannot, in general, minimize bias and variance simultaneously. There is a bias-variance tradeoff such that bias typically decreases when variance increases and vice versa. Data set size usually interacts with bias and variance and in turn affects error. Since differences between samples are expected to decrease with increasing sample size, differences between models formed from those samples are expected to decrease and hence variance is expected to decrease. Therefore, the bias proportion of error may be higher on large data sets than on small data sets and the variance proportion of error may be higher on small data sets than on large data sets. Consequently, low bias algorithms may have advantage in error on large data sets and low variance algorithms may have advantage in error on small data sets (Brain and Webb 2002).

Zheng and Webb (2005) compare eight semi-naive Bayesian methods with naive Bayes. These methods are BSE, FSS, TAN, SP-TAN, AODE, NBTree, LBR, and BSEJ. NBTree, SP-TAN, and BSEJ have relatively high training time complexity, while LBR has high classification time complexity. BSEJ has very high space complexity. NBTree and BSEJ have very low bias and high variance. Naive Bayes and AODE have very low variance. AODE has a significant advantage in error over other semi-naive Bayesian algorithms tested, with the exceptions of LBR and SP-TAN. It achieves a lower error for more data sets than LBR and SP-TAN without SP-TAN's high training time complexity and LBR's high test time complexity. Subsequent researches (Cerquides and Mántaras 2005; Zheng and Webb 2006) show that MAPLMG and SR can in practice significantly improve both classification accuracy and the precision of conditional probability estimates of AODE. However, MAPLMG imposes very high training time overheads on AODE, while SR imposes no extra training time overheads and only modest test time overheads on AODE.

Within the prevailing computational complexity constraints, we suggest using the lowest bias semi-naive Bayesian method for large training data and lowest variance semi-naive Bayesian method for small training data. An appropriate tradeoff between bias and variance should be sought for intermediate size training data. For extremely small data, naive Bayes may be superior and for large data NBTree and BSEJ may be more appealing options if their computational complexity satisfies the computational constraints of the application context. AODE achieves very low variance, relatively low bias and low training time and space complexity. MAPLMG and SR further enhance AODE by substantially reducing bias and error and improving probability prediction with modest time complexity. Consequently, they may prove competitive over a considerable range of classification tasks. Furthermore, MAPLMG may excel if the primary consideration is attaining the highest possible classification accuracy and SR may have an advantage if one wishes efficient classification.

Cross-References

▶ Averaged One-Dependence Estimators
▶ Bayesian Network
▶ Naïve Bayes

Recommended Reading

Ayer M, Brunk HD, Ewing GM, Reid WT, Silverman E (1955) An empirical distribution function for sampling with incomplete information. Ann Math Stat 26(4):641–647

Brain D, Webb GI (2002) The need for low bias algorithms in classification learning from large data sets. In: Proceedings of the sixteenth European conference on principles of data mining and knowledge discovery. Springer, Berlin, pp 62–73

Cerquides J, Mántaras RLD (2005) Robust Bayesian linear classifier ensembles. In: Proceedings of the sixteenth European conference on machine learning, Porto, pp 70–81

Frank E, Hall M, Pfahringer B (2003) Locally weighted naive Bayes. In: Proceedings of the nineteenth conference on uncertainty in artificial intelligence, Acapulco. Morgan Kaufmann, San Francisco, pp 249–256

Friedman N, Geiger D, Goldszmidt M (1997) Bayesian network classifiers. Mach Learn 29(2):131–163

Gama J (2003) Iterative bayes. Theor Comput Sci 292(2):417–430

Keogh EJ, Pazzani MJ (1999) Learning augmented Bayesian classifiers: a comparison of distribution-based and classification-based approaches. In: Proceedings of the international workshop on artificial intelligence and statistics, Fort Lauderdale, pp 225–230

Kittler J (1986) Feature selection and extraction. In: Young TY, Fu KS (eds) Handbook of pattern recognition and image processing. Academic Press, New York

Kohavi R (1996) Scaling up the accuracy of naive-Bayes classifiers: a decisiontree hybrid. In: Proceedings of the second international conference on knowledge discovery and data mining, Portland, pp 202–207

Pazzani MJ (1996) Constructive induction of Cartesian product attributes. In: ISIS: information. statistics and induction in science, Melbourne. World Scientific, Singapore, pp 66–77

Rissanen J (1978) Modeling by shortest data description. Automatica 14:465–471

Sahami M (1996) Learning limited dependence Bayesian classifiers. In: Proceedings of the second international conference on knowledge discovery in databases. AAAI Press, Menlo Park, pp 334–338

Webb GI, Pazzani MJ (1998) Adjusted probability naive Bayesian induction. In: Proceedings of the eleventh Australian joint conference on artificial intelligence, Sydney. Springer, Berlin, pp 285–295

Webb GI, Boughton J, Wang Z (2005) Not so naive Bayes: aggregating onedependence estimators. Mach Learn 58(1):5–24

Webb GI, Boughton J, Zheng F, Ting KM, Salem H (2012) Learning by extrapolation from marginal to full-multivariate probability distributions: Decreasingly naive Bayesian classification. Mach Learn 86(2):233–272

Zadrozny B, Elkan C (2002) Transforming classifier scores into accurate multiclass probability estimates. In: Proceedings of the eighth international conference on knowledge discovery and data mining, Edmonton. ACM Press, New York, pp 694–699

Zhang NL, Nielsen TD, Jensen FV (2004) Latent variable discovery in classification models. Artif Intell Med 30(3):283–299

Zheng Z, Webb GI (2000) Lazy learning of Bayesian rules. Mach Learn 41(1):53–84

Zheng F, Webb GI (2005) A comparative study of semi-naive Bayes methods in classification learning. In: Proceedings of the fourth australasian data mining conference, Sydney, pp 141–156

Zheng F, Webb GI (2006) Efficient lazy elimination for averaged-one dependence estimators. In: Proceedings of the twenty-third international conference on machine learning. ACM Press, New York, pp 1113–1120

Zheng F, Webb GI, Suraweera P, Zhu L (2012) Subsumption Resolution: An Efficient and Effective Technique for Semi-Naive Bayesian Learning. Mach Learn 87(1):93–125

Semi-supervised Learning

Xiaojin Zhu
University of Wisconsin-Madison, Madison,
WI, USA

Synonyms

Co-training; Learning from labeled and unlabeled data; Transductive learning

Definition

Semi-supervised learning uses both labeled and unlabeled data to perform an otherwise ▸ supervised learning or ▸ unsupervised learning task.

In the former case, there is a distinction between inductive semi-supervised learning and transductive learning. In inductive semi-supervised learning, the learner has both labeled training data $\{(x_i, y_i)\}_{i=1}^{l} \overset{iid}{\sim} p(\mathbf{x}, y)$ and unlabeled training data $\{\mathbf{x}_i\}_{i=l+1}^{l+u} \overset{iid}{\sim} p(\mathbf{x})$, and learns a predictor $f : \mathcal{X} \mapsto \mathcal{Y}, f \in \mathcal{F}$, where $\mathcal{F}$ is the hypothesis space. Here $\mathbf{x} \in \mathcal{X}$ is an input instance, $y \in \mathcal{Y}$ its target label (discrete for ▸ classification or continuous for ▸ regression), $p(\mathbf{x}, y)$ the unknown joint distribution and $p(\mathbf{x})$ its marginal, and typically $l \ll u$. The goal is to learn a predictor that predicts future test data better than the predictor learned from the labeled training data alone. In transductive learning, the setting is the same except that one is solely interested in the predictions on the unlabeled training data $\{\mathbf{x}_i\}_{i=l+1}^{l+u}$, without any intention to generalize to future test data.

In the latter case, an unsupervised learning task is enhanced by labeled data. For example, in semi-supervised clustering (a.k.a. ▸ constrained clustering) one may have a few must-links (two instances must be in the same cluster) and cannot-links (two instances cannot be in the same cluster) in addition to the unlabeled instances to be clustered; in semi-supervised ▸ dimensionality reduction one might have the target low-dimensional coordinates on a few instances.

This entry will focus on the former case of learning a predictor.

Motivation and Background

Semi-supervised learning is initially motivated by its practical value in learning faster, better, and cheaper. In many real world applications, it is relatively easy to acquire a large amount of unlabeled data $\{\mathbf{x}\}$. For example, documents can be crawled from the Web, images can be obtained from surveillance cameras, and speech can be collected from broadcast. However, their corresponding labels $\{y\}$ for the prediction task, such as sentiment orientation, intrusion detection, and phonetic transcript, often requires slow human annotation and expensive laboratory experiments. This labeling bottleneck results in a scarce of labeled data and a surplus of unlabeled data. Therefore, being able to utilize the surplus unlabeled data is desirable.

Recently, semi-supervised learning also finds applications in cognitive psychology as a computational model for human learning. In human categorization and concept forming, the environment provides unsupervised data (e.g., a child watching surrounding objects by herself) in addition to labeled data from a teacher (e.g., Dad points to an object and says “bird!”). There is evidence that human beings can combine labeled and unlabeled data to facilitate learning.

The history of semi-supervised learning goes back to at least the 1970s, when self-training, transduction, and Gaussian mixtures with the expectation-maximization (EM) algorithm first emerged. It enjoyed an explosion of interest since the 1990s, with the development of new algorithms like co-training and transductive support vector machines, new applications in natural language processing and computer vision, and new theoretical analyses. More discussions can be found in section 1.1.3 in Chapelle et al. (2006).

Theory

Unlabeled data $\{\mathbf{x}_i\}_{i=l+1}^{l+u}$ by itself does not carry any information on the mapping $\mathcal{X} \mapsto \mathcal{Y}$. How can it help us learn a better predictor $f : \mathcal{X} \mapsto \mathcal{Y}$? Balcan and Blum pointed out in 2009 that the key lies in an implicit ordering of $f \in \mathcal{F}$ induced by the unlabeled data. Informally, if the implicit ordering happens to rank the target predictor f^* near the top, then one needs less labeled data to learn f^*. This idea will be formalized later on using PAC learning bounds. In other contexts, the implicit ordering is interpreted as a prior over $\mathcal{F}$ or as a regularizer.

A semi-supervised learning method must address two questions: what implicit ordering is induced by the unlabeled data, and how to algorithmically find a predictor near the top of this implicit ordering and fits the labeled data well. Many semi-supervised learning methods have been proposed, with different answers to these two questions (Abney 2007; Chapelle et al. 2006; Seeger 2001; Zhu and Goldberg 2009). It is impossible to enumerate all methods in this entry. Instead, we present a few representative methods.

Generative Models

This semi-supervised learning method assumes the form of joint probability $p(\mathbf{x}, y \mid \theta) = p(y \mid \theta)p(\mathbf{x} \mid y, \theta)$. For example, the class prior distribution $p(y \mid \theta)$ can be a multinomial over $\mathcal{Y}$, while the class conditional distribution $p(\mathbf{x} \mid y, \theta)$ can be a multivariate Gaussian in $\mathcal{X}$ (Castelli and Cover 1995; Nigam et al. 2000). We use $\theta \in \Theta$ to denote the parameters of the joint probability. Each θ corresponds to a predictor f_θ via Bayes rule:

$$\begin{aligned} f_\theta(\mathbf{x}) &\equiv \mathrm{argmax}_y\, p(y|\mathbf{x}, \theta) \\ &= \mathrm{argmax}_y \frac{p(\mathbf{x}, y|\theta)}{\sum_{y'} p(\mathbf{x}, y'|\theta)}. \end{aligned}$$

Therefore, $\mathcal{F} = \{f_\theta : \theta \in \mathit{\Theta}\}$.

What is the implicit ordering of f_θ induced by unlabeled training data $\{\mathbf{x}_i\}_{i=l+1}^{l+u}$? It is the large to small ordering of log likelihood of θ on unlabeled data:

$$\log p\left(\{\mathbf{x}_i\}_{i=l+1}^{l+u} \middle| \theta\right) = \sum_{i=l+1}^{l+u} \log\left(\sum_{y\in\mathcal{Y}} p(\mathbf{x}_i, y|\theta)\right).$$

The top ranked f_θ is the one whose θ (or rather the generative model with parameters θ) best fits the unlabeled data. Therefore, this method assumes that the form of the joint probability is correct for the task.

To identify the f_θ that both fits the labeled data well and ranks high, one maximizes the log likelihood of θ on both labeled and unlabeled data:

$$\begin{aligned} &\mathrm{argmax}_\theta \log p(\{\mathbf{x}_i, y_i\}_{i=1}^{l}|\theta) \\ &\quad + \lambda \log p(\{\mathbf{x}_i\}_{i=l+1}^{l+u}|\theta), \end{aligned}$$

where λ is a balancing weight. This is a non-concave problem. A local maximum can be found with the EM algorithm, or other numerical optimization methods. (See also, ▸ generative learning.)

Semi-supervised Support Vector Machines

This semi-supervised learning method assumes that the decision boundary $f(\mathbf{x}) = 0$ is situated in a low-density region (in terms of unlabeled data) between the two classes $y \in \{-1, 1\}$ (Joachims 1999; Vapnik 1998). Consider the following hat loss function on an unlabeled instance $\mathbf{x}$:

$$\max(1 - |f(\mathbf{x})|, 0),$$

which is positive when $-1 < f(x) < 1$, and zero outside. The hat loss thus measures the violation in (unlabeled) large margin separation between f and $\mathbf{x}$. Averaging over all unlabeled training instances, it induces an implicit ordering from small to large over $f \in \mathcal{F}$:

$$\frac{1}{u} \sum_{i=l+1}^{l+u} \max(1 - |f(\mathbf{x})|, 0).$$

The top ranked f is one whose decision boundary avoids most unlabeled instances by a large margin.

To find the f that both fits the labeled data well and ranks high, one typically minimizes the following objective:

$$\operatorname{argmin}_f \frac{1}{l}\sum_{i=1}^{l}\max(1-y_i(\mathbf{x}_i),0)$$

$$+\lambda_1\|f\|^2+\lambda_2\frac{1}{u}\sum_{i=l+1}^{l+u}\max(1-|f(\mathbf{x})|,0),$$

which is a combination of the objective for supervised support vector machines, and the average hat loss. Algorithmically, the optimization problem is difficult because the hat loss is non-convex. Existing solutions include semi-definite programming relaxation, deterministic annealing, continuation method, concave-convex procedure (CCCP), stochastic gradient descent, and Branch and Bound. (See also ▶ support vector machines.)

Graph-Based Models

This semi-supervised learning method assumes that there is a graph $G = \{V, E\}$ such that the vertices V are the labeled and unlabeled training instances, and the undirected edges E connect instances i, j with weight w_{ij} (Blum and Chawla 2001; Zhu et al. 2003; Belkin et al. 2006). The graph is sometimes assumed to be a random instantiation of an underlying manifold structure that supports $p(\mathbf{x})$. Typically, w_{ij} reflects the proximity of $\mathbf{x}_i$, $\mathbf{x}_j$. For example, the Gaussian edge weight function defines $w_{ij} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/\sigma^2)$. As another example, the kNN edge weight function defines $w_{ij} = 1$ if $\mathbf{x}_i$ is within the k nearest neighbors of $\mathbf{x}_j$ or vice versa, and $w_{ij} = 0$ otherwise. Other commonly used edge weight functions include ε-radius neighbors, b-matching, and combinations of the above.

Large w_{ij} implies a preference for the predictions $f(\mathbf{x}_i)$ and $f(\mathbf{x}_j)$ to be the same. This can be formalized by the graph energy of a function f:

$$\sum_{i,j=1}^{l+u} w_{ij}(f(\mathbf{x}_i)-f(\mathbf{x}_j))^2.$$

The graph energy induces an implicit ordering of $f \in \mathcal{F}$ from small to large. The top ranked function is the smoothest with respect to the graph (in fact, it is any constant function). The graph energy can be equivalently expressed using the so-called unnormalized graph Laplacian matrix. Variants including the normalized Laplacian and the powers of these matrices.

To find the f that both fits the labeled data well and ranks high (i.e., being smooth on the graph or manifold), one typically minimizes the following objective:

$$\operatorname{argmin}_f \frac{1}{l}\sum_{i=1}^{l} c(f(\mathbf{x}_i), y_i) + \lambda_1\|f\|^2$$

$$+\lambda_2\sum_{i,j=1}^{l+u} w_{ij}(f(\mathbf{x}_i)-f(x_j))^2,$$

where $c(f(\mathbf{x}), y)$ is a convex loss function such as the hinge loss or the squared loss. This is a convex optimization problem with efficient solvers.

Co-training and Multiview Models

This semi-supervised learning method assumes that there are multiple, different learners trained on the same labeled data, and these learners agree on the unlabeled data. A classic algorithm is co-training (Blum and Mitchell 1998). Take the example of web page classification, where each web page $\mathbf{x}$ is represented by two subsets of features, or "views" $\mathbf{x} = \langle \mathbf{x}^{(1)}, \mathbf{x}^{(2)}\rangle$. For instance, $\mathbf{x}^{(1)}$ can represent the words on the page itself, and $\mathbf{x}^{(2)}$ the words on the hyperlinks (on other web pages) pointing to this page. The co-training algorithm trains two predictors: $f^{(1)}$ on $\mathbf{x}^{(1)}$ (ignoring the $\mathbf{x}^{(2)}$ portion of the feature) and $f^{(2)}$ on $\mathbf{x}^{(2)}$, both initially from the labeled data. If $f^{(1)}$ confidently predicts the label of an unlabeled instance $\mathbf{x}$, then the instance-label pair $(\mathbf{x}, f^{(1)}(\mathbf{x}))$ is added to $f^{(2)}$'s labeled training data, and vice versa. Note this promotes $f^{(1)}$ and $f^{(2)}$ to predict the same on $\mathbf{x}$. This repeats so that each view teaches the other. Multiview models generalize co-training by utilizing more than two predictors, and relaxing the requirement of having separate views (Sindhwani et al. 2005).

In either case, the final prediction is obtained from a (confidence weighted) average or vote among the predictors.

To define the implicit ordering on the hypothesis space, we need a slight extension. In general, let there be m predictors $f^{(1)}, \ldots, f^{(m)}$. Now let a hypothesis be an m-tuple of predictors $\langle f^{(1)}, \ldots, f^{(m)} \rangle$. The disagreement of a tuple on the unlabeled data can be defined as

$$\sum_{i=l+1}^{l+u} \sum_{u,v=1}^{m} c(f^{(u)}(\mathbf{x}_1), f^{(v)}(\mathbf{x}_i)),$$

where $c()$ is a loss function. Typical choices of $c()$ are the 0–1 loss for classification, and the squared loss for regression. Then the disagreement induces an implicit ordering on tuples from small to large.

It is important for these m predictors to be of diverse types, and have different ▶ inductive biases. In general, each predictor $f^{(u)}, u = 1 \ldots m$ may be evaluated by its individual loss function $c^{(u)}$ and regularizer $\Omega^{(u)}$. To find a hypothesis (i.e., m predictors) that fits the labeled data well and ranks high, one can minimize the following objective:

$$\underset{\langle f^{(1)},\ldots f^{(m)} \rangle}{\operatorname{argmin}} \sum_{u=1}^{m} \left(\frac{1}{l} \sum_{i=1}^{l} c^{(u)}(f^{(u)}(\mathbf{x}_i), y_i) + \lambda_1 \Omega^{(u)}(f^{(u)}) \right)$$
$$+ \lambda_2 \sum_{i=l+1}^{l+u} \sum_{u,v=1}^{m} c(f^{(u)}(\mathbf{x}_i), f^{(v)}(\mathbf{x}_i)).$$

Multiview learning typically optimizes this objective directly. When the loss functions and regularizers are convex, numerical solution is relatively easy to obtain. In the special cases when the loss functions are the squared loss, and the regularizers are squared ℓ_2 norms, there is a closed form solution. On the other hand, the co-training algorithm, as presented earlier, optimizes the objective indirectly with the iterative procedure. One advantage of co-training is that the algorithm is a wrapper method, in that it can use any "blackbox" learners $f^{(1)}$ and $f^{(2)}$ without the need to modify the learners.

A PAC Bound for Semi-supervised Learning

Previously, we presented several semi-supervised learning methods, each induces an implicit ordering on the hypothesis space using the unlabeled training data, and each attempts to find a hypothesis that fit the labeled training data well as well as rank high in that implicit ordering. We now present a theoretical justification on why this is a good idea. In particular, we present a uniform convergence bound by Balcan and Blum (Theorem 11 in Balcan and Blum 2009). Alternative theoretical analyses on semi-supervised learning can be found by following the recommended reading.

First, we introduce some notations. Consider the 0–1 loss for classification. Let $c^* : \mathcal{X} \mapsto \{0, 1\}$ be the unknown target function, which may not be in $\mathcal{F}$. Let err $(f) = E_{\mathbf{x}\sim p}[f(\mathbf{x}) \neq c^*(\mathbf{x})]$ be the true error rate of a hypothesis f, and $\widehat{\text{err}}(f) = \frac{1}{l}\sum_{i=1}^{l} f(\mathbf{x}_i) \neq c^*(\mathbf{x}_i)$ be the empirical error rate of f on the labeled training sample. To characterize the implicit ordering, we defined an "unlabeled error rate" $\text{err}_{unl}(f) = 1 - E_{\mathbf{x}\sim}p[\chi(f, \mathbf{x})]$, where the *compatibility function* $\mathcal{X} : \mathcal{F} \times \mathcal{X} \mapsto [0, 1]$ measures how "compatible" f is to an unlabeled instance $\mathbf{x}$. As an example, in semi-supervised support vector machines, if $\mathbf{x}$ is far away from the decision boundary produced by f, then $\chi(f, \mathbf{x})$ is large; but if $\mathbf{x}$ is close to the decision boundary, $\chi(f, \mathbf{x})$ is small. In this example, a large $\text{err}_{unl}(f)$ then means that the decision boundary of f cuts through dense unlabeled data regions, and thus f is undesirable for semi-supervised learning. In contrast, a small

$\mathrm{err}_{unl}(f)$ means that the decision boundary of f lies in a low density gap, which is more desirable. In theory, the implicit ordering on $f \in \mathcal{F}$ is to sort $\mathrm{err}_{unl}(f)$ from small to large. In practice, we use the empirical unlabeled error rate $\widehat{\mathrm{err}}_{\mathrm{unl}}(f) = 1 - \frac{1}{u}\sum_{i=l+1}^{l+u} \mathcal{X}(f, \mathbf{x}_i)$.

Our goal is to show that if an $f \in \mathcal{F}$ "fits the labeled data well and ranks high," then f is almost as good as the best hypothesis in $\mathcal{F}$. Let $t \in [0,1]$. We first consider the best hypothesis f_t^* in the subset of $\mathcal{F}$ that consists of hypotheses whose unlabeled error rate is no worse than t : $f_t^* = \mathrm{argmin}_{f'\mathcal{F},\mathrm{err}_{\mathrm{unl}}(f')\le t}\mathrm{err}(f')$. Obviously, $t = 1$ gives the best hypothesis in the whole $\mathcal{F}$. However, the nature of the guarantee has the form $\mathrm{err}(f) \le \mathrm{err}(f_{t^*}) + \mathrm{EstimationError}(t) + c$, where the EstimationError term increases with t. Thus, with $t = 1$ the bound can be loose. On the other hand, if t is close to 0, EstimationError(t) is small, but $\mathrm{err}(f_{t^*})$ can be much worse than $\mathrm{err}(f_{t=1}^*)$. The bound will account for the optimal t.

We introduce a few more definitions. Let $\mathcal{F}(f) = \{f' \in \mathcal{F} : \widehat{\mathrm{err}}_{unl}(f') \le \widehat{\mathrm{err}}_{unl}(f)\}$ be the subset of $\mathcal{F}$ with empirical error no worse than that of f. As a complexity measure, let $[\mathcal{F}(f)]$ be the number of different partitions of the first l unlabeled instances $\mathbf{x}_{l+1}\ldots\mathbf{x}_{2l}$, using $f \in \mathcal{F}(f)$. Finally, let $\hat{\epsilon}(f) = \sqrt{\frac{24}{l}\log(8[\mathcal{F}(f)])}$. Then we have the following agnostic bound (meaning that c^* may not be in $\mathcal{F}$, and $\widehat{\mathrm{err}}_{\mathrm{unl}}(f)$ may not be zero for any $f \in \mathcal{F}$):

Theorem 1 *Given l labeled instances and sufficient unlabeled instances, with probability at least 1 – δ, the function*

$$f = \mathrm{argmin}_{f' \in \mathcal{F}} \widehat{err}(f') + \hat{\epsilon}(f')$$

satisfies the guarantee that

$$err(f) \le \min_t(err(f_t^*)) + 5\sqrt{\frac{\log(8/\delta)}{l}}.$$

If a function f fits the labeled data well, it has a small $\widehat{\mathrm{err}}(f)$. If it ranks high, then $\mathcal{F}(f)$ will be a small set, consequently $\hat{\epsilon}(f)$ is small. The argmin operator identifies the best such function during training. The bound account for the minimum of all possible t tradeoffs. Therefore, we see that the "lucky" case is when the implicit ordering is good such that $f_{t=1}^*$, the best hypothesis in $\mathcal{F}$, is near the top of the ranking. This is when semi-supervised learning is expected to perform well. Balcan and Blum also give results addressing the key issue of how much *unlabeled* data is needed for $\widehat{\mathrm{err}}_{\mathrm{unl}}(f)$ and $\mathrm{err}_{\mathrm{unl}}(f)$ to be close for all $f \in \mathcal{F}$.

Applications

Because the type of semi-supervised learning discussed in this entry has the same goal of creating a predictor as supervised learning, it is applicable to essentially any problems where supervised learning can be applied. For example, semi-supervised learning has been applied to natural language processing (word sense disambiguation (Yarowsky 1995), document categorization, named entity classification, sentiment analysis, machine translation), computer vision (object recognition, image segmentation), bioinformatics (protein function prediction), and cognitive psychology. Follow the recommended reading for individual papers.

Future Directions

There are several directions to further enhance the value semi-supervised learning. First, we need guarantees that it will outperform supervised learning. Currently, the practitioner has to manually choose a particular semi-supervised learning method, and often manually set learning parameters. Sometimes, a bad choice that does not match the task (e.g., modeling each class with a Gaussian when the data does not have this distribution) can make semi-supervised learning worse than supervised learning. Second, we need methods that benefit from unlabeled when l, the size of labeled data, is large. It has been widely observed that the gain over supervised learning is the largest when l is small, but diminishes as l increases. Third, we need good ways to combine semi-supervised learning and ▸ active learning. In natural learning systems such as humans, we routinely observe unlabeled input, which often naturally leads to questions. And finally, we need methods that can efficiently process massive unlabeled data, especially in an ▸ online learning setting.

Cross-References

- ▸ Active Learning
- ▸ Classification
- ▸ Constrained Clustering
- ▸ Dimensionality Reduction
- ▸ Online Learning
- ▸ Regression
- ▸ Supervised Learning
- ▸ Unsupervised Learning

Recommended Reading

Abney S (2007) Semisupervised learning for computational linguistics. Chapman & Hall/CRC, Florida

Balcan M-F, Blum A (2009) A discriminative model for semi-supervised learning. J ACM

Belkin M, Niyogi P, Sindhwani V (2006) Manifold regularization: a geometric framework for learning from labeled and unlabeled examples. J Mach Learn Res 7:2399–2434

Blum A, Chawla S (2001) Learning from labeled and unlabeled data using graph mincuts. In: Proceedings of the 18th international conference on machine learning. Morgan Kaufmann, San Francisco, pp 19–26

Blum A, Mitchell T (1998) Combining labeled and unlabeled data with co-training. In: COLT: proceedings of the workshop on computational learning theory. ACM, New York, pp 92–100

Castelli V, Cover T (1995) The exponential value of labeled samples. Pattern Recogn Lett 16(1):105–111

Chapelle O, Zien A, Schölkopf B (eds) (2006) Semi-supervised learning. MIT Press, Cambridge

Joachims T (1999) Transductive inference for text classification using support vector machines. In: Proceedings of the 16th international conference on machine learning. Morgan Kaufmann, San Francisco, pp 200–209

Nigam K, McCallum AK, Thrun S, Mitchell T (2000) Text classification from labeled and unlabeled documents using EM. Mach Learn 39(2/3):103–134

Seeger M (2001) Learning with labeled and unlabeled data. Technical report, University of Edinburgh, Edinburgh

Sindhwani V, Niyogi P, Belkin M (2005) A co-regularized approach to semi-supervised learning with multiple views. In: Proceedings of the 22nd ICML workshop on learning with multiple views, Bonn

Vapnik V (1998) Statistical learning theory. Wiley, New York

Yarowsky D (1995) Unsupervised word sense disambiguation rivaling supervised methods. In: Proceedings of the 33rd annual meeting of the association for computational linguistics, Cambridge, pp 189–196

Zhu X, Ghahramani Z, Lafferty J (2003) Semi-supervised learning using Gaussian fields and harmonic functions. In: The 20th international conference on machine learning (ICML), Washington, DC

Zhu X, Goldberg AB (2009) Synthesis lectures on artificial intelligence and machine learning. In: Introduction to semi-supervised learning. Morgan & Claypool, San Rafael

Semi-supervised Text Processing

Ion Muslea
Language Weaver, Inc., Marina del Rey, CA,
USA

Synonyms

Learning from labeled and unlabeled data; Transductive learning

Definition

In contrast to supervised and unsupervised learners, which use solely labeled or unlabeled examples, respectively, semi-supervised learning systems exploit both labeled and unlabeled examples. In a typical semi-supervised framework, the system takes as input a (small) training set of labeled examples and a (larger) working set of unlabeled examples; the learner's performance is evaluated on a test set that consists of unlabeled examples. Transductive learning is a particular case of semi-supervised learning in which the working set and the test set are identical.

Semi-supervised learners use the unlabeled examples to improve the performance of the system that could be learned solely from labeled

S

data. Such learners typically exploit – directly or indirectly – the distribution of the available unlabeled examples. Text processing is an ideal application domain for semi-supervised learning because the abundance of text documents available on the Web makes it impossible for humans to label them all. We focus here on two related types of text processing tasks that were heavily studied in the semi-supervised framework: text classification and text ▶ Clustering.

Motivation and Background

In most applications of machine learning, collecting large amounts of labeled examples is an expensive, tedious, and error-prone process. In contrast, one may often have cheap or even free access to large amounts of unlabeled examples. For example, for text classification, which is the task of classifying text documents into categories such as politics, sports, entertainment, etc., one can easily crawl the Web and download billions of Web pages; however, manually labeling all these documents according to the taxonomy of interest is an extremely expensive task.

The key idea in semi-supervised learning is to complement a small amount of labeled data by a large number of unlabeled examples. Under certain conditions, the unlabeled examples can be mined for knowledge that will allow the semi-supervised learner to build a system that performs better than one learned solely from the labeled data. More precisely, semi-supervised learners assume that the learning model matches the structure of the application domain. If this is the case, the information extracted from the unlabeled data can be used to *guide* the search towards the optimal solution (e.g., by modifying or re-ranking the learned hypotheses); otherwise, the unlabeled examples may *hurt* rather than help the learning process (Cozman et al. 2003).

For the sake of concision and clarity, we have had to make several compromises in terms of the algorithms and the applications presented here. Given the vastness of the field of text processing, we have decided to focus only on the two related tasks of text classification and text clustering. They are the most studied text processing applications within the field of machine learning; furthermore, virtually all the main types of semi-supervised algorithms were applied to these two tasks. This decision has two main consequences. First, we do not consider many other text processing tasks, such as information extraction, natural language parsing, or base noun–phrase identification; for these we refer the interested reader to Muslea et al. (2006). Second, we discuss and cite approaches that were applied to text classification or clustering there is however, alone an excellent survey by Zhu (2005) covering seminal work on semi-supervised learning that was not applied to text processing.

Structure of the Learning System

Generative Models

The early work on semi-supervised text categorization (Nigam et al. 2000) was based primarily on generative models (see ▶ generative learning). Such approaches make two major assumptions: (1) the data is generated by a mixture model, and (2) there is a correspondence between the components of the mixture and the classes of the application domain. Intuitively, if these assumptions hold, the unlabeled examples become instrumental in identifying the mixture's components, while the labeled examples can be used to label each individual component.

The iterative approach proposed by Nigam et al. (2000) is based on The EM Algorithm and works as follows. First, the labeled examples are used to learn an initial classifier, which is used to probabilistically label all unlabeled data; then the newly labeled examples are added to the training set. Finally, a new classifier is learned from all the data, and the entire process is repeated till convergence is reached (or, alternatively, till the number of iterations is fixed).

Nigam et al. (2000) noticed that, in practice, the two above-mentioned assumptions about the generative model may not hold; in order to deal with this problem, the authors propose two extensions of their basic approach. First, they allow each class to be generated by multiple mixture components. Second, they introduce a weighting factor that adjusts the contribution of the unlabeled examples; this factor is tuned during the learning process so that the influence of the unlabeled examples correlates with the degree in which the data distribution is consistent with the mixture model.

The same general framework can also be applied to the related task of text clustering. In the clustering framework, the learner is not concerned with the actual label of an example; instead, it tries to find a partitioning of the examples in clusters that are similar *respect to* a predefined objective function. For example, Seeded-KMeans (Basu et al. 2002) is a semi-supervised text clustering algorithm that uses the few available labeled examples to seed the search for the data clusters. In order to optimize the target objective function, Seeded-KMeans uses an EM algorithm on a mixture of Gaussians.

Discriminative Approaches

▸ Support vector machines (SVMs) (Joachims 1999) are particularly well suited for text classification because of their ability to deal with high-dimensional input spaces (each word in the corpus is a feature) and sparse feature-value vectors (any given document contains only a small fraction of the corpus vocabulary). SVMs are called maximum margin classifiers because they minimize the empirical classification error by maximizing the geometric margin between the domain's positive and negative examples. Intuitively, this is equivalent to finding a discriminative decision boundary that avoids the high-density regions in the instance space.

Transductive SVMs (Joachims 1999) are designed to find an optimal decision boundary for a particular test set. More precisely, they have access to both the (labeled) training set and the unlabeled test set. Transductive SVMs work by finding a labeling of the test examples that maximizes the margin over all the examples in the training and the test set. This transductive approach has shown significant improvements over the traditional inductive SVMs, especially if the size of the training set is small.

In contrast to transductive SVMs, semi-supervised SVMs (S3VM) work in a true semi-supervised setting in which the test set is not available to the learner. A major difficulty in the S3VM framework is the fact that the resulting optimization problem is not convex, thus being sensitive to the issue of (non-optimal) local minima. CS3VMs (Chapelle et al. 2006) alleviate this problem by using a global optimization technique called continuation. On binary classification tasks CS3VMs compare favorably against other S3VM approaches, but applying it on multiclass domains is still an open problem.

Multiview Approaches

Multiview learners are a class of algorithms for domains in which the features can be partitioned in disjoint subsets (views), each of which is sufficient to learn the target concept. For example, when classifying Web pages, one can use either the words that appear in the documents or those that appear in the hyper-links pointing to them. Co-training (Blum and Mitchell 1998) is a semi-supervised, multiview learner that, intuitively, works by bootstrapping the views from each other. First, it uses the labeled examples to learn a classifier in each view. Then it applies the learned classifiers to the unlabeled data and detects the examples on which each view makes the most confident prediction; these examples are labeled by the respective classifiers and added to the (labeled) training set of the other view. The entire process is repeated for a number of iterations.

Multiview learners rely on two main assumptions, namely that the views are compatible and uncorrelated. The former requires that each example is identically labeled by the target concept in each view; the latter means that given an example's label, its description in each view are independent. In practice, both these assumptions are likely to be violated; in order to deal with the first issue, one can use the adaptive view validation algorithm (Muslea et al. 2002b), which predicts whether the views are sufficiently compatible for multiview learning.

With respect to view correlation Muslea et al. (2002a) have shown that by interleaving active and semi-supervised learning, multiview approaches become robust the view correlation. A similar idea was previously used in the generative, single-view framework: McCallum and Nigam (1998) have shown that by allowing the algorithm to (smartly) choose which examples to include in the training set, one can significantly improve over the performance of both supervised and semi-supervised learners that used randomly chosen training sets.

The main limitation of multiview learning is the requirement that the user identifies at least two suitable views. In order to cope with this problem, researchers have proposed algorithms that work in a way similar to co-training, but exploit multiple ▶ inductive biases instead of multiple views. For example, tri-training (Zhou and Li 2005) uses all domain features to train three supervised classifiers (e.g., a decision tree, a neural network, and a Naive Bayes classifier). These classifiers are then applied to each unlabeled example; if two of them agree on the example's label, they label it accordingly and add it to the third classifier's training set. A degenerate case is represented by *self-training*, which uses a single classifier that repeatedly goes through the unlabeled data and adds to its own training set, the examples on which its predictions are the most confident.

Graph-Based Approaches

The work on graph-based, semi-supervised text learning is based on the idea of representing the labeled and unlabeled examples as vertices in a graph. The edges of this graph are weighted by the pair-wise similarity between the corresponding examples, thus offering a flexible way to incorporate prior domain knowledge. With the learning task encoded in this manner, the problem to be solved becomes one of graph theory, namely finding a partitioning of the graph that agrees with the labeled examples. A major challenge for the graph-based approaches is to find a balanced partitioning of the graph (e.g., in a degenerate scenario, one can propose an unbalanced, undesirable partition in which, except for the negative examples in the training set, all other examples are labeled as positive).

One possible approach to cope with the issue on unbalanced partitions is to use randomized min-cuts (Blum et al. 2004). The algorithm starts with the original graph and repeatedly adds random noise to the weights of the edges. Then, for each modified graph, it finds a partitioning by using minimum cuts. Finally, the results from the various runs aggregated in order to create probabilistic labels for the unlabeled examples. This approach has the additional benefit of offering a measure of the confidence in each particular prediction.

The SGT algorithm (Joachims 2003) uses spectral methods to perform the graph partitioning. SGT can be seen as a transductive version of the k nearest-neighbor classifier; furthermore Joachims (2003) also show that co-training emerges as a special case of SGT. In contrast to transductive SVMs and co-training, SGT does not require additional heuristics for avoiding unbalanced graph partitionings (e.g., in the original co-training algorithm, the examples that are added to the training set after each iteration must respect the domain-dependent ratio of negative-to-positive examples).

LapSVM (Sindhwani et al. 2005) is a graph-based kernel method that uses a weighted combination a regularizer learned solely from labeled data and a graph Laplacian obtained from both the labeled and unlabeled examples. This approach allows LapSVM to perform a principled search for a decision boundary that is both consistent with the labeled examples and reflects the underlying geometry of all available data points.

Approaches that Exploit Background Knowledge

WHIRL-BG (Zelikovitz and Hirsh 2000) is an algorithm for classifying short text fragments. It uses an information integration approach that combines three different information sources: the training set, which consists of the labeled examples; the test set that WHIRL-BG must label;

and a secondary corpus that consists longer, related documents that are not labeled. Intuitively, WHIRL-BG exploits the secondary corpus as background knowledge that allows the system to link a test example to the most similar labeled training example. In other words, instead of trying to measure directly a (unreliable) similarity between two short strings (i.e., a test and a training example), the system searches for a background document that may include (a large fraction of) both strings.

HMRF-KMEANS (Basu et al. 2004) unifies the two main approaches to semi-supervised text clustering: the constraint-based one and the adaptive distance one. The former exploits user-provided background knowledge to find an appropriate partitioning of the data; for HMRF-KMEANS, the domain knowledge consists of must-link or cannot-link constraints, which specify whether two examples should or should not have the same label, respectively. The later uses a small number of labeled examples to learn a domain-specific distance measure that is appropriate for the clustering task at hand. HMRF-KMEANS can use any Bregman divergence to measure the clustering distortion, thus supporting a wide variety of learnable distances.

HMRF-KMEANS exploits the labeled examples in three main ways. First, it uses the neighborhoods induced from the constraints to initialize the cluster centroids. Second, when assigning examples to clusters, the algorithm tries to simultaneously minimize both the similarity to the cluster's centroid and the number of violated constraints. Last but not least, during the clustering process, HMRF-KMEANS iteratively reestimates the distance measure so that it takes into account both the background knowledge and the data variance.

Recommended Reading

Basu S, Banerjee A, Mooney R (2002) Semi-supervised clustering by seeding. In: Proceedings of the international conference on machine learning, Sydney, pp 19–26

Basu S, Bilenko M, Mooney R (2004) A probabilistic framework for semi-supervised clustering. In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining, Seattle, pp 59–68

Blum A, Lafferty J, Rwebangira MR, Reddy R (2004) Semi-supervised learning using randomized mincuts. In: Proceedings of the twenty-first international conference on machine learning, Banff, p 13

Blum A, Mitchell T (1998) Combining labeled and unlabeled data with co-training. In: Proceedings of the 1988 conference on computational learning theory, pp 92–100

Chapelle O, Chi M, Zien A (2006) A continuation method for semi-supervised SVMs. In: Proceedings of the 23rd international conference on machine learning. ACM Press, New York, pp 185–192

Cozman F, Cohen I, Cirelo M (2003) Semi-supervised learning of mixture models. In: Proceedings of the international conference on machine learning, Washington, DC, pp 99–106

Joachims T (1999) Transductive inference for text classification using support vector machines. In: Proceedings of the 16th international conference on machine learning (ICML-99). Morgan Kaufmann, San Francisco, pp 200–209

Joachims T (2003) Transductive learning via spectral graph partitioning. In: Proceedings of the international conference on machine learning, Washington, DC

McCallum A, Nigam K (1998) Employing EM in pool-based active learning for text classification. In: Proceedings of the 15th international conference on machine learning, Madison, pp 359–367

Muslea I, Minton S, Knoblock C (2002a) Active + semi-supervised learning = robust multi-view learning. In: The 19th international conference on machine learning (ICML-2002), Sydney, pp 435–442

Muslea I, Minton S, Knoblock C (2002b) Adaptive view validation: a first step towards automatic view detection. In: The 19th international conference on machine learning (ICML-2002), Sydney, pp 443–450

Muslea I, Minton S, Knoblock C (2006) Active learning with multiple views. J Artif Intell Res 27:203–233

Nigam K, McCallum AK, Thrun S, Mitchell TM (2000) Text classification from labeled and unlabeled documents using EM. Mach Learn 39(2/3):103–134

Sindhwani V, Niyogi P, Belkin M (2005) Beyond the point cloud: from transductive to semi-supervised learning. In: Proceedings of the 22nd international conference on machine learning, Bonn, pp 824–831

Zelikovitz S, Hirsh H (2000) Improving short text classification using unlabeled background knowledge. In: Proceedings of the 17th international conference on machine learning, Stanford, pp 1183–1190

Zhou Z-H, Li M (2005) Tri-training: exploiting unlabeled data using three classifiers. IEEE Trans Knowl Data Eng 17(11):1529–1541

Zhu X (2005) Semi-supervised learning literature survey. Technical report 1530, Department of Computer Sciences, University of Wisconsin, Madison

Sensitivity

Synonyms

Recall; True positive rate

Sensitivity is the fraction of positive examples predicted correctly by a model. See ▸ Sensitivity and Specificity, ▸ Precision and Recall.

Sensitivity and Specificity

Kai Ming Ting
Federation University, Mount Helen, VIC,
Australia

Definition

Sensitivity and specificity are two measures used together in some domains to measure the predictive performance of a classification model or a diagnostic test. For example, to measure the effectiveness of a diagnostic test in the medical domain, sensitivity measures the fraction of people with disease (i.e., positive examples) who have a positive test result; and specificity measures the fraction of people without disease (i.e., negative examples) who have a negative test result. They are defined with reference to a special case of the ▸ confusion matrix, with two classes, one designated the *positive* class and the other the *negative* class, as indicated in Table 1.

Sensitivity and Specificity, Table 1 The outcomes of classification into positive and negative classes

		Assigned class	
		Positive	Negative
Actual class	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Sensitivity is sometimes also called *true positive rate*. Specificity is sometimes also called *true negative rate*. They are defined as follows:

$$\text{Sensitivity} = \text{TP}/(\text{TP} + \text{FN})$$
$$\text{Specificity} = \text{TN}/(\text{TN} + \text{FP})$$

Instead of two measures, they are sometimes combined to provide a single measure of predictive performance as follows:

$$\text{Sensitivity} \times \text{Specificity}$$
$$= \text{TP} * \text{TN}/[(\text{TP} + \text{FN}) * (\text{TN} + \text{FP})]$$

Note that sensitivity is equivalent to ▸ recall.

Cross-References

▸ Confusion Matrix

Sentiment Analysis

▸ Sentiment Analysis and Opinion Mining

Sentiment Analysis and Opinion Mining

Lei Zhang[1] and Bing Liu[2]
[1]LinkedIn, San Francisco, CA, USA
[2]University of Illinois at Chicago, Chicago, IL, USA

Abstract

With the rapid growth of social media, sentiment analysis, also called opinion mining, has become one of the most active research areas in natural language processing. Its application is also widespread, from business services to political campaigns. This article gives an introduction to this important area and presents some recent developments.

Synonyms

Opinion extraction; Opinion mining; Sentiment analysis; Sentiment mining

Definition

Sentiment analysis or opinion mining is the computational study of people's opinions, sentiments, appraisals, attitudes, and emotions toward entities and their aspects expressed in text.

Motivation and Background

Sentiment and opinion and their related concepts, such as evaluation, appraisal, attitude, affect, emotion, and mood, are about people's subjective beliefs and feelings. They are key influencers of human behaviors. Whenever we need to make a decision, we often seek out others' opinions. This is true for both individuals and organizations.

The development of sentiment analysis coincides with the growth of *social media* (i.e., reviews, forum discussions, and blogs) on the Web. For the first time in human history, we now possess a huge volume of opinion data recorded in digital forms. These *user-generated contents* (UGC) are full of people's opinions. Mining useful knowledge from these corpora gives rise to the task of sentiment analysis. Since the early 2000s, it has been one of the most active research areas in natural language processing (NLP) (Pang and Lee 2008; Liu 2012). The research and applications have also spread from computer science to management science and social sciences because of its importance to business and society as a whole. Sentiment analysis techniques have been widely applied in practice, from business services to political campaigns.

Structure of the Task

In a nutshell, the task of sentiment analysis is to mine people's opinions and emotions from text. The term *opinion* is used as a concept represented with a quadruple (s, g, h, t) covering four components (Liu 2012): sentiment orientation s, sentiment target g opinion holder h, and time t. *Sentiment* is the underlying feeling, attitude, evaluation, or emotion associated with an opinion. Sentiment orientation can be *positive*, *negative*, or *neutral*. *Sentiment target*, also known as the *opinion target*, is an *entity* or an *aspect* of the *entity* that the sentiment has been expressed upon. *Opinion holder* is an individual or organization that holds an opinion. *Time* is when the opinion is expressed. We will discuss emotion specifically later.

We use the following camera review as an example (an ID number is associated with each sentence for easy reference):

Posted by John Smith
Date: September 10, 2011

(1) *I bought a Canon G12 camera six months ago.* (2) *I simply love it.* (3) *The picture quality is amazing.* (4) *The battery life is also long.* (5) *However, my wife thinks it is too heavy for her.*

Given the review, the task of sentiment analysis aims to extract the following opinion quadruples from sentences 2, 3, 4, and 5, respectively:

(*positive*, *Canon G12 camera*, *author*, *2011/09/10*)
(*positive*, *picture quality* of *Canon G12 camera author*, *2011/09/10*)
(*positive*, *battery life* of *Canon G12 camera*, *author*, *2011/09/10*)
(*negative*, *weight* of *Canon G12 camera, author's wife*, *2011/09/10*)

The opinion target can be an entity (*Canon G12 camera*) or an aspect of the entity (*picture quality*, *battery life*, and *weight* of the Canon G12 camera). An aspect can be *explicit* (e.g., *battery life*) or *implicit* (e.g., *weight* is indicated by *heavy*) (Hu and Liu 2004).

In many applications, it is useful to decompose opinion target to entity and aspect for more fine-grained analysis. Then, the above quadruples become the following quintuples, where GENERAL represents the entity itself (Liu 2012):

(*positive*, *Canon G12 camera*, *GENERAL*, *author*, *2011/09/10*)

S

(*positive*, *Canon G12 camera picture quality*, *author*, *2011/09/10*)
(*positive*, *Canon G12 camera*, *battery life*, *author*, *2011/09/10*)
(*negative*, *Canon G12 camera, weight, author's wife*, *2011/09/10*)

An opinion from a single opinion holder is usually not actionable in an opinion mining application. The user often needs opinions from a large number of opinion holders, which leads to opinion summary. A summary of opinions is normally constructed based on positive and negative sentiments about opinion targets, which is called aspect-based opinion summary (or feature-based opinion summary) (Hu and Liu 2004). Figure 1 shows an opinion summary generated from product reviews of Apple iPad by Google products. Generally, opinion summary needs to be quantitative, which is reflected by the proportions or the numbers of positive and negative opinions for each sentiment target or aspect.

Sentiment Analysis Methods

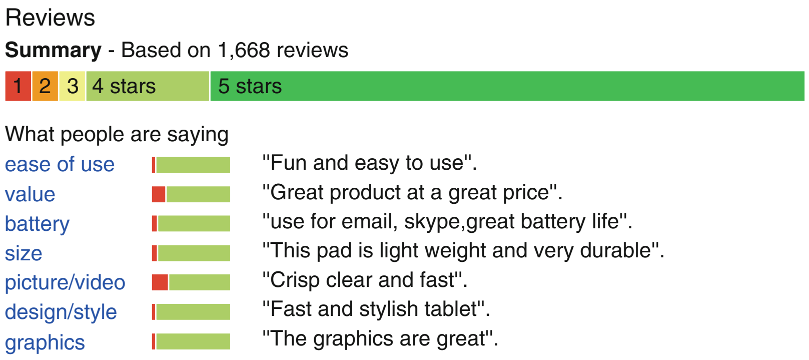
Researchers have studied sentiment analysis at three main levels of granularities: document, sentence, and aspect levels.

Document Sentiment Classification

Document sentiment classification classifies an opinion document (e.g., a product review) as expressing a positive or negative sentiment. It does not study or extract any information within the document. The task is also known as the *document-level sentiment classification*.

Document sentiment classification is commonly formulated as a supervised learning problem with two classes (*positive* and *negative*) or rating scores (e.g., 1–5 stars). Standard supervised learning methods such naïve Bayesian classification and support vector machines (SVM) can be applied for classification directly. Pang et al. (2002) first adopted those classification methods to classify movie reviews into two classes. Since this work, numerous other works have been reported. Like most supervised learning approaches, the main task of these works is to engineer a set of effective features. See Liu (2012) for an overview of this line of research.

There are also unsupervised approaches to document sentiment classification, which are mainly based on sentiment words and language patterns. It is quite clear that *sentiment words* (also called *opinion words*) that indicate positive or negative sentiments (e.g., *good* and *nice* are positive sentiment words, and *horrible* and *bad* are negative sentiment words) play an important role in sentiment classification (Turney 2002; Hu and Liu 2004; Kim and Hovy 2004). Turney (2002) proposed an unsupervised approach based on syntactic opinion patterns and Web search. Taboada et al. (2011) adapted a *lexicon-based approach* for document sentiment classification. It basically uses a set of sentiment words and phrases with appropriate scores and an aggregation scheme to aggregate the scores of


Reviews

Summary - Based on 1,668 reviews

What people are saying

ease of use "Fun and easy to use".
value "Great product at a great price".
battery "use for email, skype,great battery life".
size "This pad is light weight and very durable".
picture/video "Crisp clear and fast".
design/style "Fast and stylish tablet".
graphics "The graphics are great".

Sentiment Analysis and Opinion Mining, Fig. 1 Opinion summary for iPad from Google products the sentiment words appeared in a document to perform the classification. The lexicon-based approach was originally proposed for aspect-level and sentence-level sentiment classification (Hu and Liu 2004; Kim and Hovy 2004).

Researchers found that supervised sentiment classification is domain-sensitive, that is, a classifier trained using opinion documents from one domain performs poorly when it is applied or tested on opinion documents from another domain. The reason is that words used in different domains for expressing opinions can be different. Furthermore, the same word may mean positive in one domain but negative in another domain. Domain adaptation or transfer learning techniques have been employed to address the problem (Blitzer et al. 2007; Pan et al. 2010; Liu 2012).

Another interesting topic is cross-language sentiment classification, which focuses on using the extensive resources and tools available in English and automated translation to help build sentiment classifiers in other languages with few resources or tools (Wan 2009; Mihalcea et al. 2007). Existing research proposed three main strategies: (1) translate test documents in the target language into the source language and classify them using a source language classifier, (2) translate a source language training corpus into the target language and build a classifier in the target language, and (3) translate a sentiment lexicon in the source language to the target language and build a lexicon-based classifier in the target language.

Sentence Sentiment Classification

Sentence sentiment classification is similar to document sentiment classification as sentences can be regarded as short documents. However, sentence classification is usually harder because the information contained in a typical sentence is much less than that contained in a typical document. Furthermore, sentence sentiment classification needs to consider the neutral class (or no opinion) because there are many factual sentences that express no positive or negative opinion in an opinion document. Document classification normally does not consider the neutral class.

Document sentiment classification techniques can be naturally applied for sentence sentiment classification. Some sentence-specific approaches have also been proposed, e.g., hierarchical sequence learning model (McDonald et al. 2007) and deep learning methods (Socher et al. 2013). In addition, researchers found that different types of sentences may need different kinds of classification methods, e.g., *conditional sentences* and *integrative sentences* (Liu 2012).

For example, a conditional sentence describes implications or hypothetical situations and their consequences. Such a sentence typically contains two clauses that are dependent on each other: the condition clause and the consequent clause. Their relationship has significant impact on whether the sentence expresses a positive or negative sentiment (Narayanan et al. 2009). For example, the sentence *If someone makes a reliable car, I will buy it* expresses no sentiment toward any particular car, although it contains the positive sentiment word *reliable*. In Narayanan et al. (2009), supervised learning was used to deal with the problem using a set of linguistic features, e.g., sentiment words or phrases and their locations, part-of-speech tags of sentiment words, tense patterns, and conditional connectives.

Another type of difficult sentences is the *sarcasm sentences*. Sarcasm is a sophisticated form of speech act in which the speakers or the writers say or write the opposite of what they mean. In the context of sentiment analysis, it means that when one says something positive, one actually means negative, and vice versa. Sarcastic sentences are very difficult to deal with because commonsense knowledge and discourse analysis are often required to recognize them (Tsur et al. 2010; Riloff et al. 2013).

At the sentence level, another popular research problem is to identify *subjectivity* and *objective sentence*s. Subjective expressions express opinions, appraisals, evaluations, allegations, desires, beliefs, suspicions, speculations, or stances (Wiebe et al. 2004). Some of these concepts indicate positive or negative sentiments. Some of them do not, e.g., *I want to buy a* *camera that can take good photos* which is a subjective sentence but does not express a positive or negative sentiment about anything. Objective sentences state facts. However, we should note that objective sentences can imply positive or negative sentiments of their authors because there are desirable facts and undesirable facts (Zhang and Liu 2011). For example, the sentence *I bought the mattress a week ago and a valley has formed in the middle* states a fact, but the fact is undesirable. It thus implies a negative opinion about the quality of the mattress.

Aspect Sentiment Classification

Aspect-level classification classifies or determines sentiment on individual targets, which both the document-level and the sentence-level classification do not do because no sentiment target is involved at these two coarse levels of analysis. However, in applications, one often needs to know opinion targets. Without targets, any positive or negative sentiment is of limited use. For example, the sentence *trying out Chrome because Firebox works poorly* expresses a negative sentiment. But if we do not know that the negative sentiment is toward Firefox, not Chrome, the sentiment is of little use and can even be misleading. Many sentences also have mixed sentiments, e.g., *The performance of the car is great but the price is too high.* Aspect sentiment classification should find the opinion on the *performance* aspect of the car to be positive and the opinion on the *price* aspect of the car to be negative. In short, aspect sentiment classification determines sentiments expressed on entities and aspects of entities, which gives more useful information than document or sentence classification.

Although supervised learning can be applied, the kinds of features used for document and sentence sentiment classification are no longer sufficient or appropriate. The key reason is that those features do not consider (or are independent of) opinion targets and are thus unable to determine to which target an opinion refers. To remedy this problem, opinion target needs to be considered in learning. Two kinds of approaches have been proposed. The first one is to generate a set of features that are dependent on each opinion target in the sentence, e.g., weighing features based on their distances to a target. The second approach is to check the application scope of each sentiment expression to determine whether it covers the target in the sentence. For example, in the sentence *Apple is doing very well in this bad economy,* the sentiment word *bad*'s application scope covers only *economy*, not *Apple*. Current supervised learning methods mainly use the first approach but also have a flavor of the second approach (Jiang et al. 2011).

The lexicon-based approach can be employed as well. It computes the sentiment orientation on a target in a sentence by using a *sentiment aggregation function* that takes into account the distances of the sentiment expressions (sentiment words or phrases) and the opinion target in the sentence and/or by exploiting syntactic relationships of sentiment expressions and opinion targets to find the *application scope* of each sentiment expression. At the high level, the lexicon-based approach works as follows: it uses (1) a lexicon of sentiment expressions including sentiment words, phrases, idioms, and composition rules, (2) a set of rules for handling different language constructs (e.g., sentiment shifters and but-clauses) and different types of sentences, and (3) a sentiment aggregation function or a set of sentiment and target relationships derived from the parse tree to determine the sentiment orientation on each target (Ding et al. 2008; Liu 2012).

Comparative Sentences

Unlike a regular opinion sentence, a comparative sentence expresses a relation based on similarities or differences of more than one entity. In English, comparisons are usually conveyed using the *comparative* or *superlative* forms of adjectives or adverbs, e.g., *The picture quality of Canon cameras is better than that of Sony cameras.* To mine comparative opinions, aspect sentiment analysis is necessary because it does not make much sense to classify a comparative sentence as expressing a positive, negative, or neutral sentiment. See

Jindal and Liu (2006) and Liu (2015) for more details.

Supervised Learning vs. Lexicon-Based Approach

The key advantage of supervised learning for sentiment classification is that it can automatically learn from all kinds of features for classification through optimization. Most of these features are difficult to use by a lexicon-based method. However, supervised learning depends on the training data, which needs to be manually labeled for each domain. A shortcoming of the approach is that a supervised classifier trained from the labeled data in one domain often does not work in another domain. Thus, for each domain, new training data needs to be labeled, which is time consuming. Another shortcoming is that it is hard to learn things that do not occur frequently. The lexicon-based approach is able to avoid these issues to some extent and has been shown to perform well in a large number of applications. Its main advantage is domain independence, that is, it can be applied to any domain without manual labeling of a large amount of training data as required in the supervised learning approach. The lexicon-based method is also flexible in the sense that the system can be easily extended and improved. If an error occurs, the user simply corrects some existing rules and/or adds new rules to the system's rule base. However, the lexicon-based approach also has its disadvantages. It needs heavy investments in time and effort to build the initial knowledge base of lexicon, patterns, and rules. Furthermore, although the lexicon-based approach is supposed to be domain independent, some additional work is still needed to take care of the idiosyncrasies of each domain. The main issue is that it is quite hard to deal with domain-dependent or context-dependent sentiment words and phrases (see below and Liu 2015).

Aspect and Entity Extraction

The task of aspect and entity extraction is to identify and extract opinion targets (aspect or entity) from opinion documents. Since aspect extraction and entity extraction are closely related tasks, ideas and methods proposed for aspect extractions can also be shared with the entity extraction task. Much of the existing research focused on aspect extraction. Current aspect extraction methods can be roughly grouped into four categories: mining frequent noun phrases, utilizing syntactic relations of sentiment words and their targets, and applying supervised sequence learning models and topic modeling. All these approaches are used in practice.

Finding Frequent Noun Phrases

Since people often use the same words when they comment on the same product aspects, Hu and Liu (2004) makes use of this observation to mine aspects by simply finding frequent nouns and noun phrases in reviews using frequent itemset mining (Agrawal and Srikant 1994). Those more frequent noun phrases are also likely to be more important aspects because people usually comment on those more important aspects more frequently.

Exploiting Syntactic Relations of Sentiment and Target

It was observed in Hu and Liu (2004) that adjective sentiment words often modify (or describes) noun aspects (e.g., *great picture*). Hu and Liu (2004) used such relations to identify aspects that are hard to find by the frequency-based method above. Zhuang et al. (2006) formulated the idea based on the dependency grammar and extracted aspect and sentiment word pairs from movie reviews using a set of dependency relations. Qiu et al. (2011) developed the idea further and proposed an algorithm called *double propagation* (DP). DP uses a set of manually compiled dependency rules derived from some dependency relations to identify both aspects and sentiment words simultaneously through a bootstrapping process. These methods are all based on the idea that opinion always has target, and there are often syntactic relations that connect sentiment words and targets in a sentence. Thus, sentiment words can be recognized by identified aspects, and aspects can be identified by known sentiment words. The extracted sentiment words and aspects are utilized to identify new sentiment words and new aspects, which are used again to extract more sentiment words and aspects. This is the approach used in the DP method. Recently, this method was improved with automated rule selection (Liu et al. 2015) and word alignment from the machine translation research (Liu et al. 2013).

Applying Supervised Sequence Learning Models

Sequence learning models such as Hidden Markov models (HMM) and conditional random fields (CRF) are widely used in information extraction. They are thus also used for aspect extraction. Aspect extraction can be regarded as a sequence labeling task since entity, aspect, and opinion expressions are often interdependent and occur in a sequence in a sentence. Jin and Ho (2009) utilized lexicalized HMM to extract product aspects and opinion expressions from reviews. Different from traditional HMM, they integrated linguistic features such as part of speech and lexical patterns into HMM. Jakob and Gurevych (2010) utilized CRF to extract opinion aspects from opinion sentences.

Topic Modeling

Topic models such as PLSA (probabilistic latent semantic analysis) and LDA (latent Dirichlet allocation) have been popularly used to mine hidden topics from the document corpora. In the context of aspect extraction, aspects are basically topics in topic modeling. Mei et al. (2007) proposed a model for extracting both aspects and sentiment words. Titov and McDonald (2008) pointed out that global topic models such as PLSA and LDA might not be suitable for detecting aspects from reviews. To tackle this problem, they proposed some multigrain topic models to discover aspects, which models two distinct types of topics: global topics and local topics. Lin and He (2009) proposed a joint topic-sentiment model, which extended LDA by adding a sentiment layer. It detects sentiment and aspect simultaneously from the corpus. Further works along a similar line have been done in Brody and Elhadad (2010), Wang et al. (2010), Zhao et al. (2010), and Jo and Oh (2011). Recently, two new types of models were proposed*:* *knowledge-based models* (Mukherjee and Liu 2012) which can exploit prior domain knowledge to produce better results and *lifelong topic models* (Chen and Liu 2014) which exploit the big data to automatically mine prior knowledge to be used in the modeling process.

Sentiment Lexicon

It is quite clear that sentiment words are instrumental for sentiment analysis. Positive sentiment words are used to express some desired states, while negative ones are used to express some undesired states. Examples of positive sentiment words are *beautiful*, *wonderful*, and *good*. Examples of negative sentiment words are *bad*, *poor*, and *terrible*. Apart from individual words, there are also sentiment phrases and idioms. To compile a sentiment word list or lexicon, two approaches have been studied: dictionary-based approach and corpus-based approach.

Dictionary-Based Approach

This approach is based on bootstrapping using a small set of seed sentiment words and an online dictionary, e.g., WordNet or thesaurus. The strategy is to first collect a small set of sentiment words manually with known orientations and then to grow this set by searching in the WordNet or an online thesaurus to find their synonyms and antonyms. The newly found words are added to the seed list. The next iteration starts. The iterative process stops when no more new words are found (Hu and Liu 2004; Kim and Hovy 2004; Kamps et al. 2004).

Corpus-Based Approach

The corpus-based approach relies on syntactic patterns and also a seed list of sentiment words to find other sentiment words in a large corpus. One of the key ideas was proposed in Hatzivassiloglou and McKeown (1997). The technique uses the set of seed sentiment words and a set of linguistic constraints or conventions on connectives to identify additional sentiment words and their orientations. One of the constraints is about the conjunction AND, which says that conjoined adjectives usually have the same sentiment orientation. For example, if *beautiful* is known to be positive, we can infer that *spacious* is also positive from the sentence *This car is beautiful and spacious*. Rules or constraints are also designed for other connectives, OR, BUT, EITHER-OR, and NEITHER-NOR. This constraint is called *sentiment consistency*. Kanayama and Nasukawa (2006) and Ding et al. (2008) expanded this approach to intra-sentential and inter-sentential sentiment consistency. Ding et al. (2008) further showed that the same word may indicate positive sentiment in one sentence context but negative sentiment in another sentence context. For example, in the domain of car reviews, the word "quiet" expresses opposite sentiments or opinions in the following two sentences: *This car is very quiet* (positive) and *The audio system in the car is very quiet* (negative). The authors proposed to consider both the sentiment word and the aspect together in determining the sentiment orientation of the sentiment word. To determine the sentiment orientation of the pair, the above sentiment consistency idea is still used. In a similar vein, Choi and Cardie (2009) studied the problem of adapting a general-purpose sentiment lexicon to a specific domain of application.

Sentiment Analysis of Emotions

Emotions are human feelings. They are similar and also different from opinions. An opinion expresses an evaluation or appraisal about some objects, whereas an emotion expresses a human inner feeling. Human beings have many different types of emotions. However, there is still no agreement among researchers on how many kinds of emotions there are and what they are. According to Parrott (2001), humans have six basic emotions: joy, love, anger, fear, sadness, and surprise. Existing sentiment analysis of emotions is focused on classification of emotion types expressed in sentences. Both supervised learning and lexicon-based approaches have been attempted by researchers.

In supervised learning, Alm et al. (2005) classified the emotional affinity of sentences in the narrative domain of children's fairy tales. The features are not the traditional word n-grams but fourteen groups of Boolean features about each sentence and its context in the document. The classes are only two: neutral and emotional. In Mohammad (2012), a Twitter data set was annotated with emotion types based on emotion words or hashtags in Twitter posts. The author then performed classification of emotions using SVM with binary features that capture the presence or absence of unigrams and bigrams. Additional references can be found in Liu (2015).

In the lexicon-based approach, Yang et al. (2007) first constructed an emotion lexicon and then performed emotion classification at the sentence level using the lexicon. To construct the emotion lexicon, the proposed algorithm uses only sentences with a single user-provided emoticon. For each word, it computes a collocation (or association) strength of the word with each emoticon using a measure similar to pointwise mutual information (PMI). Those top-scoring words are very likely to indicate different types of emotions. For emotion classification of sentences, it experimented with two approaches: the lexicon-based approach and the supervised learning approach. For supervised learning, only the top k emotion words were used as features.

Summary

This article gave a brief introduction to sentiment analysis. Interested readers can refer to Liu (2015) for an in-depth and comprehensive coverage of the topic. Sentiment analysis is a highly challenging research problem with almost unlimited applications. It has been one of the most active research areas in natural language processing for many years. Although significant progresses have been made and numerous industrial systems have been built, the problem remains to be very difficult. The accuracy results in many cases are still unsatisfactory. However, the practical application needs and technical challenges will keep the field vibrant and lively for years to come.

Cross-References

▶ Opinion Stream Mining

Recommended Reading

Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proceedings of the international conference on very large databases (VLDB-1994), Santiago de Chile

Alm CO, Roth D, Sproat R (2005) Emotions from text: machine learning for text-based emotion prediction. In: Proceedings of conference on human language technology and empirical methods in natural language processing (HLT/EMPNLP-2005), Vancouver

Blitzer J, Dredze M, Pereira F (2007) Biographies, bollywood, boom-boxes and blenders: domain adaptation for sentiment classification. In: Proceedings of the annual meeting of the association for computational linguistics (ACL-2007), Prague

Brody S, Elhadad S (2010) An unsupervised aspect-sentiment model for online reviews. In: Proceedings of the annual conference of the North American chapter of the ACL (NAACL-2010), Los Angeles

Chen Z, Liu B (2014) Topic modeling using topics from many domains, lifelong learning and big data. In: Proceedings of the international conference on machine learning (ICML-2014), Beijing

Choi Y, Cardie C (2009) Adapting a polarity lexicon using integer linear programming for domain-specific sentiment classification. In: Proceedings of conference on empirical methods in natural language processing (EMNLP-2009), Singapore

Ding X, Liu B, Yu PS (2008) A holistic lexicon-based approach to opinion mining. In: Proceedings of the conference on web search and web data mining (WSDM-2008), Palo Alto

Hatzivassiloglou V, McKeown K (1997) Predicting the semantic orientation of adjectives. In: Proceedings of the annual meeting of the association for computational linguistics (ACL-1997), Madrid

Hu M, Liu B (2004) Mining and summarizing customer reviews. In: Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining (KDD-2004), Seattle

Jakob N, Gurevych I (2010) Extracting opinion targets in a single- and cross-domain setting with conditional random fields. In: Proceedings of the conference on empirical methods in natural language processing (EMNLP-2010). MIT, Massachusetts

Jiang L, Yu M, Zhou M, Liu X, Zhao T (2011) Target-dependent twitter sentiment classification. In: Proceedings of the annual meeting of the association for computational linguistics (ACL-2011), Portland

Jin W, Ho HH (2009) A novel lexicalized HMM-Based learning framework for web opinion mining. In: Proceedings of international conference on machine learning (ICML-2009), Montreal

Jindal N, Liu B (2006) Mining comparative sentences and relations. In: Proceedings of national conference on artificial intelligence (AAAI-2006), Boston

Jo Y, Oh A (2011) Aspect and sentiment unification model for online review analysis. In: Proceedings of the conference on web search and web data mining (WSDM-2011), Hong Kong

Kamps J, Marx M, Mokken RJ, De Rijke M (2004) Using WordNet to measure semantic orientation of adjectives. In: Proceedings of international conference on language resources and evaluation (LREC-2004), Lisbon

Kanayama H, Nasukawa T (2006) Fully automatic lexicon expansion for domain-oriented sentiment analysis. In: Proceedings of conference on empirical methods in natural language processing (EMNLP-2006), Sydney

Kim S-M, Hovy E (2004) Determining the sentiment of opinions. In: Proceedings of international conference on computational linguistics (COLING-2004), University of Geneva, Geneva

Li S, Lin C, Song Y, Li Z (2010) Comparable entity mining from comparative questions. In: Proceedings of the annual meeting of the association for computational linguistics (ACL-2010), Uppsala University, Uppsala

Lin C, He Y (2009) Joint sentiment/topic model for sentiment analysis. In: Proceedings of ACM international conference on information and knowledge management (CIKM-2009), Hong Kong

Liu B (2012) Sentiment analysis and opinion mining. Morgan & Claypool, San Rafael

Liu B (2015) Sentiemnt analysis: mining opinions, sentiments, and emotions. Cambridge University Press, Cambridge

Liu K, Xu L, Zhao J (2013) Syntactic patterns versus word alignment: extracting opinion targets from online reviews. In: Proceedings of the annual meeting of the association for computational linguistics (ACL-2013), Sofia

Liu Q, Gao Z, Liu B, Zhang Y (2015) Automated rule selection for aspect extraction in opinion mining. In: Proceedings of international joint conference on artificial intelligence (IJCAI-2015), Buenos Aires

McDonald R, Hannan K, Neylon T, Wells M, Reynar J (2007) Structured models for fine-to-coarse sentiment analysis. In: Proceedings of the annual meeting of the association for computational linguistics (ACL2007), Prague

Mei Q, Ling X, Wondra M, Su H, Zhai C (2007) Topic sentiment mixture: modeling facets and opinions in weblogs. In: Proceedings of international conference on world wide web (WWW-2007), Banff

Mihalcea R, Banea C, Wiebe J (2007) Learning multilingual subjective language via cross-lingual projections. In: Proceedings of the annual meeting of the association for computational linguistics (ACL-2007), Prague

Mohammad SM (2012) #Emotional tweets. In: Proceedings of the first joint conference on lexical and computational semantics, Montreal

Mukherjee A, Liu B (2012) Aspect extraction through semi-supervised modeling. In: Proceedings of the annual meeting of association for computational linguistics (ACL-2012), Jeju Island

Narayanan R, Liu B, Choudhary A (2009) Sentiment analysis of conditional sentences. In: Proceedings of conference on empirical methods in natural language processing (EMNLP-2009), Singapore

Pan SJ, Ni X, Sun J, Yang Q, Chen Z (2010) Cross-domain sentiment classification via spectral feature alignment. In: Proceedings of international conference on world wide web (WWW-2010), Raleigh

Pang B, Lee L (2008) Opinion mining and sentiment analysis. Foundations and trends in information retrieval. Now Publishers, Hanover, MA

Pang B, Lee L, Vaithyanathan S (2002) Thumbs up?: sentiment classification using machine learning techniques. In: Proceedings of conference on empirical methods in natural language processing (EMNLP-2002), Philadelphia

Parrott WG (2001) Emotions in social psychology: essential readings. Psychology Press, Hove

Qiu G, Liu B, Bu J, Chen C (2011) Opinion word expansion and target extraction through double propagation. Comput Linguist, 37(1):9–27

Riloff E, Qadir A, Surve P, De Silva L, Gilbert N, Huang R (2013) Sarcasm as contrast between a positive sentiment and negative situation. In: Proceedings of conference on empirical methods in natural language processing (EMNLP-2013), Seattle

Socher R, Perelygin A, Wu J, Manning C, Ng A, Chuang J (2013) Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of the conference on empirical methods on natural language processing (EMNLP-2013), Seattle

Taboada M, Brooke J, Tofiloski M, Voll K, Stede M (2011) Lexicon-based methods for sentiment analysis. Comput Linguist 37(2):267–307

Titov I, McDonald R (2008) Modeling online reviews with multi-grain topic models. In: Proceedings of international conference on world wide web (WWW-2008), Beijing

Tsur O, Davidov D, Rappoport A (2010) A great catchy name: semi-supervised recognition of sarcastic sentences in online product reviews. In: Proceedings of the international AAAI conference on weblogs and social media (ICWSM-2010), Washington, DC

Turney PD (2002) Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. In: Proceedings of the annual meeting of the association for computational linguistics (ACL-2002), Philadelphia

Wan X (2009) Co-training for cross-lingual sentiment classification. In: Proceedings of the annual meeting of the ACL and the IJCNLP of the AFNLP (ACL-IJCNLP-2009), Singapore

Wang H, Lu Y, Zhai C (2010) Latent aspect rating analysis on review text data: a rating regression approach. In: Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining (KDD-2010), Washington, DC

Wiebe J, Wilson T, Bruce R, Bell M, Martin M (2004) Learning subjective language. Comput Linguist 30(3):277–308

Yang C, Lin KH-Y, Chen H-H (2007) Building emotion lexicon from weblog corpora. In: Proceedings of the annual meeting of the ACL on interactive poster and demonstration sessions, Prague

Zhang L, Liu B (2011) Identifying noun product features that imply opinions. In: Proceedings of the annual meeting of the association for computational linguistics (ACL-2011), Portland

Zhao W, Jiang J, Yan H, Li X (2010) Jointly modeling aspects and opinions with a MaxEnt-LDA hybrid. In: Proceedings of conference on empirical methods in natural language processing (EMNLP-2010). MIT, Massachusetts

Zhuang L, Jing F, Zhu X (2006) Movie review mining and summarization. In: Proceedings of ACM international conference on information and knowledge management (CIKM-2006), Arlington

Sentiment Mining

► Sentiment Analysis and Opinion Mining

Separate-and-Conquer Learning

► Covering Algorithm

Sequence Data

▶ Sequential Data

Sequential Data

Synonyms

Sequence Data

Sequential Data refers to any *data* that contain elements that are ordered into sequences. Examples include ▶ time series, DNA sequences (see ▶ biomedical informatics) and sequences of user actions. Techniques for learning from sequential data include ▶ Markov models, ▶ Conditional Random Fields and ▶ time series techniques.

Sequential Inductive Transfer

▶ Cumulative Learning

Sequential Learning

▶ Online Learning

Set

▶ Class

Shannon's Information

If a message announces an event E_1 of probability $P(E_1)$ its information content is $-\log_2 P(E_1)$. This is also its length in bits.

Shattering Coefficient

Synonyms

Growth function

Definition

The shattering coefficient $S_{\mathcal{F}}(n)$ is a function that measures the size of a function class $\mathcal{F}$ when its functions $f : \mathcal{X} \to \mathbb{R}$ are restricted to sets of points $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{X}^n$ of size n. Specifically, for each $n \in \mathbb{N}$ the shattering coefficient is the maximum size of the set of vectors $\mathcal{F}_{\mathbf{x}} = \{(f(x_1), \ldots., f(x_n)) : f \in \mathcal{F}\} \subset \mathbb{R}^n$ that can be realized for some choice of $\mathbf{x} \in \mathcal{X}^n$. That is,

$$S_{\mathcal{F}}(n) = \sup_{\mathbf{x} \in \mathcal{X}^n} |\mathcal{F}_x|.$$

The shattering coefficient of a hypothesis class $\mathcal{H}$ is used in ▶ generalization bounds as an analogue to the class's size in the finite case.

Sigmoid Calibration

▶ Classifier Calibration

Similarity Measures

Michail Vlachos
IBM Research, Zurich, Switzerland

Synonyms

Distance; Distance functions; Distance measures; Distance metrics

Introduction

The term similarity measure refers to a function that is used for comparing objects of any type. The objects can be data structures, database records, or even multimedia objects (audio, video, etc). Therefore the input of a similarity measure is two objects, and the output is, typically, a number between 0 and 1; "zero" meaning that the objects are completely dissimilar and "one" signifying that the two objects are identical. Similarity is related to distance, which is the inverse of similarity, that is, a similarity of 1 implies a distance of 0 between two objects.

Background

Similarity measures are typically used for quantifying the affinity between objects in search operations, in which the user presents an object (query) and requests other objects "similar" to the given query. Therefore, a similarity measure is a mathematical abstraction for comparing objects and it assigns a single number that indicates the affinity between the said pair of objects. The results of the search are customarily presented to the user in the order suggested by the similarity value returned. Objects with higher similarity value are presented first because they are deemed to be more relevant to the query posed by the user. For example, when searching for specific keywords on an internet search engine, internet pages that are more relevant/similar to the query posed are presented first. The selection of the proper similarity function is an important parameter in many applications, including ▸ instance-based learning, ▸ clustering, and ▸ anomaly detection.

Most similarity measures attempt to model (imitate) the human notion of similarity between objects. If a similarity function resembles very closely the similarity ranking between objects as returned by a human, then it is considered successful. This, however, is also where the difficulty lies because in general similarity is something that is very subjective.

Consider the case where a user poses the keyword query "crane" at a search engine while searching for images. The results returned would contain images of machinery, birds, or even origami creations. This is because when the similarity measure used is solely based on textual information then all such images are indeed proper answers to the query. If one were also interested in the semantics of an image, then perhaps additional features such as texture, color, or shape could have been used. Therefore, to define an effective similarity measure, one first has to extract the proper *object features* and then evaluate the similarity using an appropriate distance function.

Classes of Similarity Functions

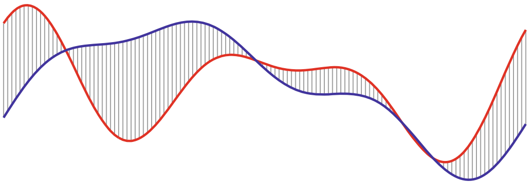
There are two major classes of similarity functions: metric and nonmetric functions. For a function d to be a metric, it has to satisfy all of the following three properties for any objects X, Y, Z:

1. $d(X,Y) = 0$ iff $X = Y$ (identity axiom)
2. $d(X,Y) = d(Y,X)$ (symmetry axiom)
3. $d(X,Y) + d(Y,Z) \geq d(X,Z)$ (triangle inequality)

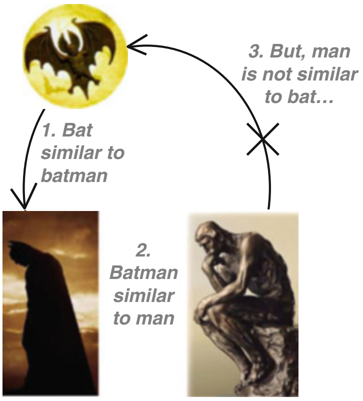
Metric similarity functions are very widely used in search operations because they support the triangle inequality. The triangle inequality can help prune much of the search space by eliminating objects from examination that are guaranteed to be distant to the given query (Agrawal et al. 1993; Zezula 2005). The most frequently used metric similarity function is the Euclidean distance. For two objects X and Y that are characterized by set of n features $X = (x_1, x_2, \ldots, x_n)$ and similarly $Y = (y_1, y_2, \ldots, y_n)$, the Euclidean distance is defined as

$$D = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

S



Similarity Measures, Fig. 1 Mapping achieved by the Euclidean distance between time-series data



Similarity Measures, Fig. 2 Nonmetric similarity that disobeys the triangle inequality

If we represent the objects X and Y as an ordered sequence of their features, we can visualize the linear mapping achieved by the Euclidean distance in Fig. 1.

Nonmetric similarity measures resemble more closely the human notion of similarity by allowing a more flexible matching between the objects examined, for example, by allowing nonlinear mappings or even by accommodating occlusion of points or features. The human visual system is in general considered to be nonmetric. Nonmetric measures typically disobey the triangle inequality. For example, consider the following nonmetric relationship: "Batman" is similar to "man," and "bat" is also similar to "batman," but this does not imply that "bat" is similar to "man." This is illustrated in Fig. 2.

Examples for Time-Series Data

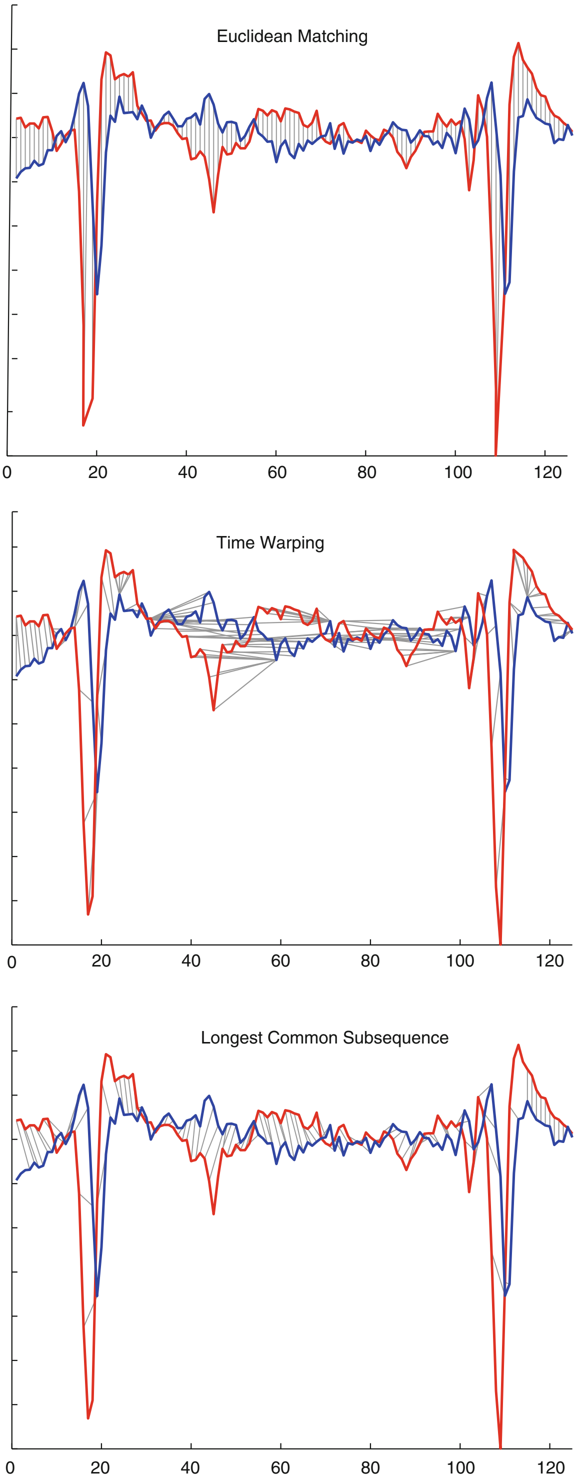
Consider the case of time-series data. Widely used nonmetric similarity functions are the warping distance and the longest common subsequence (LCSS). The warping distance (also known as dynamic time warping or DTW) has been used very extensively in the past in voice-recognition tasks because of its ability to perform compression or decompression of the features, allowing flexible nonlinear mappings. In Fig. 3, we depict the outcome of the measures for time-series data mentioned above. The Euclidean distance performs a rigid linear mapping of points, the DTW can perform nonlinear one-to-many mappings, and the LCSS constructs a one-to-one nonlinear mapping.

Recently, similarity metrics based on information theory, and in specific, on Kolmogorov complexity, have been presented (Li 2004; Keogh 2004) and can also be considered as *compression-based* measures. A very simple and easily implementable version of a compression-based distance is

$$d_c(X,Y) = \frac{C(XY)}{C(X)+C(Y)}$$

where $C(X)$ is the compressed size (bytes) of X given a certain compression algorithm. The distance will be close to 1 if X and Y are dissimilar and less than 1 if X and Y are related. Therefore we exploit the fact that if X and Y are "similar," they should compress equally well (approximately same number of bytes) when considered either separately or together because the compression dictionaries will be similar when the two objects are related. In summary, the choice of similarity measure is highly dependent on the application at hand. The practitioner should also closely consider on which object features the similarity measure will be applied. Ultimately, the combination of both feature selection and similarity measure will define the quality of a search process.

Similarity Measures, Fig. 3 Comparison of Euclidean, warping, and longest common subsequence measures



Cross-References

▶ Dimensionality Reduction
▶ Feature Selection

Recommended Reading

Agrawal R, Faloutsos C, Swami A (1993) Efficient similarity search in sequence databases. In: Proceeding of the FODO, Chicago

Keogh E, Lonardi S, Ratanamahatana A (2004) Towards parameter-free data mining. In: Proceedings of the SIGKDD, Seattle

Li M, Chen X, Li X, Ma B, Vitanyi PMB (2004) The similarity metric. IEEE Trans Inf Theory 50(12):3250–3264

Zezula P, Amato G, Dohnal V, Batko M (2005) Similarity search: the metric space approach. Advances in database systems. Springer, New York

Simple Bayes

▶ Naïve Bayes

Simple Recurrent Network

Risto Miikkulainen
Department of Computer Science, The University of Texas at Austin, Austin, TX, USA

Synonyms

Elman network; Feedforward recurrent network

Definition

The simple recurrent network is a specific version of the ▶ backpropagation neural network that makes it possible to process sequential input and output (Elman 1990). It is typically a three-layer network where a copy of the hidden layer activations is saved and used (in addition to the actual input) as input to the hidden layer in the next time step. The previous hidden layer is fully connected to the hidden layer. Because the network has no recurrent connections per se (only a copy of the activation values), the entire network (including the weights from the previous hidden layer to the hidden layer) can be trained with the backpropagation algorithm as usual. It can be trained to read a sequence of inputs into a target output pattern, to generate a sequence of outputs from a given input pattern, or to map an input sequence to an output sequence (as in predicting the next input). Simple recurrent networks have been particularly useful in ▶ time series prediction, as well as in modeling cognitive processes, such as language understanding and production.

Recommended Reading

Elman JL (1990) Finding structure in time. Cognit Sci 14:179–211

SMT

▶ Statistical Machine Translation

Solution Concept

A criterion specifying which locations in the search space are solutions and which are not. In designing a coevolutionary algorithm, it is important to consider whether the solution concept implemented by the algorithm (i.e., the set of individuals to which it can converge) corresponds with the intended solution concept.

Solving Semantic Ambiguity

▶ Word Sense Disambiguation

SOM

▶ Self-Organizing Maps

Sort

▶ Class

Spam Detection

▶ Text Mining for Spam Filtering

Specialization

Specialization is the converse of ▶ generalization. Thus, if h_1 is a generalization of h_2 then h_2 is a specialization of h_1.

Cross-References

▶ Generalization
▶ Induction
▶ Learning as Search
▶ Logic of Generality
▶ Subsumption

Specificity

Synonyms

True negative rate

Specificity is the fraction of negative examples predicted correctly by a model. See ▶ Sensitivity and Specificity.

Spectral Clustering

▶ Graph Clustering
▶ K-Way Spectral Clustering

Speedup Learning

Alan Fern
Science, Oregon State University, Corvallis, OR, USA

Definition

Speedup learning is a branch of machine learning that studies learning mechanisms for speeding up problem solvers based on problem-solving experience. The input to a speedup learner typically consists of observations of prior problem-solving experience, which may include traces of the problem solver's operations and/or solutions to solve the problems. The output is knowledge that the problem solver can exploit to find solutions more quickly than before learning without seriously effecting the solution quality. The most distinctive feature of speedup learning, compared with most branches of machine learning, is that the learned knowledge does not provide the problem solver with the ability to solve new problem instances. Rather, the learned knowledge is intended solely to facilitate faster solution times compared to the solver without the knowledge.

Motivation and Background

Much of the work in computer science and especially artificial intelligence aims at developing practically efficient problem solvers for combinatorially hard problem classes such as automated planning, logical and probabilistic reasoning, game playing, constraint satisfaction, and combinatorial optimization. While it is often straightforward to develop optimal problem solvers for these problems using brute-force, exponential-time search procedures, it is generally much more difficult to develop solvers that are efficient across a wide range of problem instances. The main motivation behind speedup learning is to create adaptive problem solvers that can learn patterns from problem-solving experience that can be exploited for efficiency gains. Such adaptive solvers have the potential to significantly outperform traditional static solvers by specializing their behavior to the characteristics of a single problem instance or to an entire class of related problem instances. The exact form of knowledge and learning mechanism is tightly tied to the problem class and the problem-solver architecture.

Most branches of machine learning, such as ▶ supervised classification, aim to learn fundamentally new problem-solving capabilities that are not easily programmed by hand even when ignoring efficiency issues – for example, learning to recognize handwritten digits. Speedup learning is distinct in that it is typically applied in situations where hand-coding an optimal, but inefficient, problem solver is straightforward – for example, solving satisfiability problems. Rather, learning is aimed exclusively at finding solutions in a more practical time frame.

Work in speedup learning grew out of various subfields of artificial intelligence and more generally computer science. An early example, from automated planning, involved learning knowledge for speeding up the original STRIPS planner (Fikes et al. 1972) via the learning of triangle tables or macros that could later be exploited by the problem solver. Throughout the 1980s and early 1990s, there was a great deal of additional work on speedup learning in the area of automated planning as overviewed in Minton (1993) and Zimmerman and Kambhampati (2003).

Another major source of speedup learning research has originated from the areas of AI search and constraint satisfaction. Many of the ▶ intelligent backtracking mechanisms from these areas, which are critical to perform, can be viewed as speedup learning techniques (Kambhampati 1998) where knowledge is learned, while solving a problem instance that better informs later search decisions. Such methods have also come out of the area of logic programming (Kumar and Lin 1988), where search efficiency plays a central role.

In addition, various branches of AI have developed speedup learning approaches based on learning-improved heuristic evaluation functions. Samuel's checker player (Samuel 1959) was one such early example, where learned evaluation functions allowed for the performance of deep game tree search to be approximated by a shallower, less expensive search.

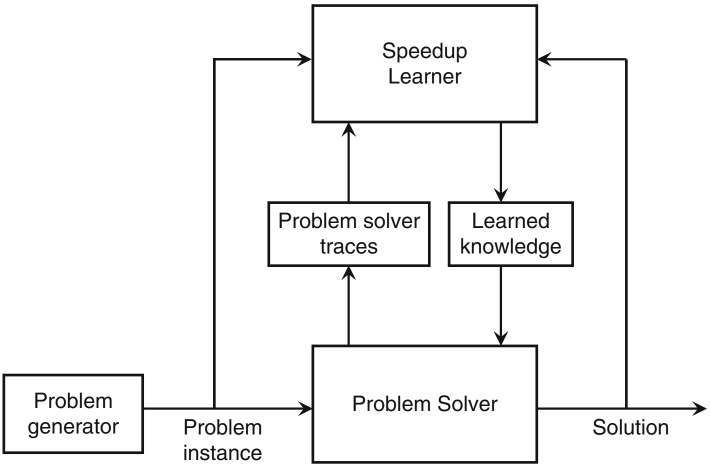
Structure of Learning System

Figure 1 shows a generic diagram of a speedup learning system. The main components are the problem solver and the speedup learner. The role of the problem solver is to receive problem instances from a problem generator and to produce solutions for those instances. For example, problem solvers might include constraint-satisfaction engines, automated planners, or A^* search. The role of the speedup learner is to produce knowledge that the problem solver can use to improve its solution time. The input to the speedup learner, which is analyzed in order to produce the knowledge, can include one or more of the following data sources: (1) the input problem instances, (2) traces of the problem solver's decisions while solving the input problems, and (3) solutions to solved problems.

Clearly there is a large space of possible speedup learning systems that result from different problem solvers, forms of learned knowledge, learning methods, and intended mode of applicability. Some of the main dimensions are described in the following section in which speedup learning approaches can be characterized. Examples of typical learners that span this space are provided, noting that the examples are far from an exhaustive list.

Dimensions of Speedup Learning

Intra-problem Versus Inter-problem Speedup. Intra-problem speedup learning is when knowledge is learned during the solution of the current problem instance and is only applicable to speeding up the solution of the current instance. After a solution is found, the knowledge is discarded as it is not applicable for the future instances. Inter-problem speedup learning is when the learned knowledge is applicable not only to the problem(s) it was learned on but also to new problems



Speedup Learning, Fig. 1 Schematic diagram of a speedup learning system. The problem solver receives problem instances from a problem generator and produces solutions. The speedup learner can observe the input problem instances, traces of the problem solver while solving the problem instances, and sometimes also the solutions to previously solved problem instances. The speedup learner outputs knowledge that can be used by the problem solver to speed up its solution time either on the current problem instance (intra-problem speedup) or future related instances (inter-problem speedup)

to be encountered in the future. In this sense, the learned knowledge can be viewed as a generalized knowledge about how to find solutions more quickly for an entire class of problems.

Typically in the inter-problem learning, the problem generator produces instances that are related in some way and, thus, share common structure that can be learned from the earlier instances and exploited when solving the later instances. Rather intra-problem speedup learners treat each problem instance as completely distinct from the rest. Also note that inter-problem learners have the potential to benefit from the analysis of solutions to previous problem instances. Rather, intra-problem learners are unable to use this source of information, since once the current problem is solved, no further learning is warranted.

Types of Learned Knowledge. Most problem solvers can be viewed as search procedures, which is the view that will be taken when characterizing the various forms of learned knowledge in speedup learning. Four types of commonly used knowledge are listed below, noting that this is far from an exhaustive list. First, *pruning constraints* are the sets of constraints on search nodes that signal when certain branch of the search space can be safely pruned. Second, *macro operators* (macros) are sequences of search operators that are typically useful when executed in order. Problem solvers can often utilize macros in order to decrease the effective solution depth of the search space by treating macros as additional search operators. It is important that the decrease in effective depth is enough to compensate for the increase in number of operators, which increases the search complexity. Third, *search-control rules* are the sets of rules that typically test the current problem-solving state and suggest problem-solving actions such as rejecting, selecting, or preferring a particular search operator. In the extreme case, learned search-control rules can completely remove the need for search. Fourth, *heuristic evaluation functions* are used to measure the quality of a particular search node. Learning-improved heuristics can result in better directed search behavior.

Deductive Versus Inductive Learning. ▶ Deductive learning refers to a learning process for which the learned knowledge can be deductively proven to be correct. For example, in the case of learned pruning constraints, a deductive learning mechanism would provide a guarantee that the pruning was sound in the sense that the optimality of the problem solver would be unaffected. ▶ Inductive learning mechanisms rather are statistical in nature and typically do not produce knowledge with associated deductive guarantees. Rather, inductive methods focus on finding statistical regularities that are typically useful, though perhaps not correct in all cases. For example, an inductive learner may discover patterns that are strongly correlated to pruning opportunities, though these patterns may have a small probability of leading to unsound pruning.

In cases where one must guarantee a sound and complete problem solver, deductive learning approaches are always applicable, though their utility depends on the particular application. In certain cases, inductively learned knowledge can also be utilized in a way that does not affect the correctness of the problem solver. For example, inductively learned search-control rules that assert preferences, rather than prune nodes from the search, do not lead to incompleteness. Traditionally, the primary disadvantage of deductive learning, compared with inductive learning, is that the inductive methods typically produce knowledge that generalizes to a wider range of situations than deductive methods. In addition, deductive learning methods are often more costly in terms of learning time as they rely on expensive deductive reasoning mechanisms. Naturally, a number of speedup learning systems exist that utilize a combination of inductive and deductive learning techniques.

Examples of Intra-problem Speedup Learning

Much of the speedup learning work arising from research in AI search and constraint satisfaction falls into the intra-problem paradigm. The most common forms of learning are deductive and are based on computing explanations of "search failures" that occur during the solution of a particular problem. Here a search failure typically corresponds to a point where the problem solver must backtrack. By computing and forming such failure explanations, the problem solver is typically able to avoid similar types of failures in the future by detecting that a search path will lead to failure without fully exploring that path. ▶ Nogood learning is a very successful, and commonly used, example of the general failure-explanation approach (Schiex and Verfaillie 1994). Nogoods are combinations of variable values that lead to search failures. By computing and recording nogoods, it is possible to immediately prune search states that consider those value combinations. There are many variations of nogood learning, with different techniques utilizing different approaches to analyzing search failures to extract general nogoods.

Another example of the failure-explanation approach, which is commonly utilized in satisfiability solvers, is ▶ clause learning. The idea is similar to nogood learning. When a failure occurs during the systematic search, a proof of the failure is constructed and analyzed to extract implied constraints, or clauses, that the solution must satisfy. These learned clauses are then added to the set of clauses of the original satisfiability problem and in later search trigger early pruning when they, or their consequences, are violated. Efficient implementations of this idea have led to huge gains in satisfiability solvers. In addition, it has been shown theoretically that clause learning can improve solution times by an exponential factor (Beame et al. 2004).

Inductive techniques for learning heuristic evaluation functions have also been investigated in the intra-problem speedup paradigm. Here we discuss just two such approaches, where in both cases the key idea is to observe the problem solver and extract training examples that can be used to learn an accurate evaluation function. A particularly successful example of this approach is the STAGE system (Boyan and Moore 1998) for solving combinatorial optimization problems such as traveling salesman and circuit layout. The problem-solving architecture used by STAGE is based on repeated random restarts of a fast hill-climbing local optimizer, which, when given an initial configuration of the combinatorial object, performs a greedy search to a local minimum configuration. The speedup learning mechanism for STAGE is to learn an approximate function that maps initial configurations to the performance of the local optimizer when started at that configuration. Note that on each restart of the problem solver, the learning component gets a training example that can be used to improve the function. The problem solver uses the learned function in order to select promising configurations from which to restart, rather than choosing randomly. In particular, STAGE attempts to restart from a configuration that optimizes the learned function, which is the predicted best starting point for the hill climber. This overall approach has shown impressive performance gains in a number of combinatorial optimization domains.

As a second example of inductive learning of heuristics in the intra-problem paradigm, there has been work within the more traditional problem-solving paradigm of best-first search (Sarkar et al. 1998). Here the speedup learner observes the sequence of search nodes traversed by the problem solver. For any pair of nodes observed to be on the same search path, the learner creates a training example in an attempt to train a heuristic to better predict the distance between those two nodes. Ideally, this updated heuristic function better reflects the distance from nodes in the search queue to the goal node of the current problem instance and, hence, results in improved search performance.

Examples of Inter-problem Speedup Learning

Much of the work on inter-problem speedup learning came out of AI planning research, where researchers have long studied learning approaches for speeding up planners. Speedup in planning is focused in this chapter, noting that similar ideas have also been pursued in other research areas such as constraint satisfaction. For a collection and survey of work on speedup in planning, see Minton (1993) and Zimmerman and Kambhampati (2003). Typically in this work, one is interested in learning knowledge for an entire planning domain, which is a collection of problems that share the same set of actions. The Blocksworld is a classic example of such a planning domain. After experiencing and solving a number of problems from a target domain, such as the Blocksworld, the learned knowledge is then used to speed up performance on new problems from the same domain.

There have been a number of deductive learning approaches to speed up learning in planning, which are traditionally cited as ▸ explanation-based learning (EBL) approaches (Minton et al. 1989). EBL for AI planning is strongly related to the failure-explanation approaches developed for CSPs as characterized nicely by Kambhampati (1998). There are two main differences between the inter-problem EBL work in planning and the intra-problem EBL approaches for CSPs. First, EBL approaches in planning produce more general explanations that are applicable not only in the problem in which they were learned but also new problems. This is often made possible by introducing variables in the place of specific objects into the explanations derived from a particular problem. This allows the explanations to apply to contexts in new problems that share similar structure but involve different objects. The second difference is that inter-problem EBL approaches in planning often produce explanations of successes and not just of failures. These positive explanations are not possible in the context of intra-problem speedup since the intra-problem learner is only interested in solving a single problem.

Despite the relatively large effort invested in inter-problem EBL research, the best approaches typically did not consistently lead to significant gains and even hurt performance in many cases. A primary way that EBL can hurt performance is by learning too many explanations, which results in the problem solver spending too much time simply evaluating the explanations at the cost of reducing the number of search nodes considered. This problem is commonly referred to as the EBL utility problem (Minton 1988) as it is difficult to determine which explanations have high enough utility to be worth keeping.

In addition to EBL, there has also been work on inductive mechanisms for acquiring search-control rules to speed up AI planners. Typically, statistical learning mechanisms are used to find common patterns that can distinguish between good and bad search decisions. As one example, Huang et al. learn action-rejection and selection rules based on the solutions to planning problems from a common domain (Huang et al. 2000). The learned rules were then added as constraints to the constraint satisfaction engine, which served to guide the solver to solution plans more quickly. Another approach, which has been studied at a theoretical and empirical level, is to learn heuristic functions to guide a bounded search process (Xu and Fern 2009), in particular, breadth-first beam search. Results in a number of planning domains demonstrate significant improvements over planners that do not incorporate a learning component. One other class of approach is based on attempting to learn knowledge that removes the need for a problem solver altogether, in particular, to learn a reactive policy for quickly selecting actions in any given state of the environment. Such policies can be learned via statistical techniques by simply trying to learn an efficient function that maps planning states to the actions selected by the planner. Despite its simplicity, this approach has demonstrated considerable success (Khardon 1999) and has also been characterized at a theoretical level (Tadepalli and Natarajan 1996).

Cross-References

▸ Explanation-Based Learning

Recommended Reading

Beame P, Kautz H, Sabharwal A (2004) Towards understanding and harnessing the potential of clause learning. J Artif Intell Res 22:319–351

Boyan JA, Moore AW (1998) Learning evaluation functions for global optimization and boolean satisfiability. In: National conference on artificial intelligence, Madison. AAAI, Menlo Park, pp 3–10

Fikes R, Hart P, Nilsson N (1972) Learning and executing generalized robot plans. Artif Intell 3(1–3):251–288

Huang Y-C, Selman B, Kautz H (2000) Learning declarative control rules for constraint-based planning. In: International conference on machine learning, Stanford. Morgan Kaufmann, San Francisco, pp 415–422

Kambhampati S (1998) On the relations between intelligent backtracking and failure-driven explanation-based learning in constraint satisfaction and planning. Artif Intell 105(1–2):161–208

Khardon R (1999) Learning action strategies for planning domains. Artif Intell 113(1–2):125–148

Kumar V, Lin Y (1988) A data-dependency based intelligent backtracking scheme for prolog. J Log Program 5(2):165–181

Minton S (1988) Quantitative results concerning the utility of explanation-based learning. In: National conference on artificial intelligence, St. Paul. Morgan Kaufmann, St. Paul, pp 564–569

Minton S (ed) (1993) Machine learning methods for planning. Morgan Kaufmann, San Francisco

Minton S, Carbonell J, Knoblock CA, Kuokka DR, Etzioni O, Gil Y (1989) Explanation-based learning: a problem solving perspective. Artif Intell 40:63–118

Samuel A (1959) Some studies in machine learning using the game of checkers. IBM J Res Dev 3(3):211–229

Sarkar S, Chakrabarti P, Ghose S (1998) Learning whiles solving problems in best first search. IEEE Trans Syst Man Cybern A Syst Hum 28(4):553–541

Schiex T, Verfaillie G (1994) Nogood recording for static and dynamic constraint satisfaction problems. Int J Artif Intell Tools 3(2):187–207

Tadepalli P, Natarajan B (1996) A formal framework for speedup learning from problems and solutions. J Artif Intell Res 4:445–475

Zimmerman T, Kambhampati S (2003) Learning-assisted automated planning: looking back, taking stock, going forward. AI Mag 24(2):73–96

Speedup Learning for Planning

▸ Explanation-Based Learning for Planning

Spike-Timing-Dependent Plasticity

A biological form of Hebbian learning where the change of synaptic weights depends on the exact timing of presynaptic and postsynaptic action potentials.

Cross-References

▸ Biological Learning: Synaptic Plasticity, Hebb Rule and Spike Timing Dependent Plasticity

Split Tests

► Online Controlled Experiments and A/B Testing

Sponsored Search

► Text Mining for Advertising

Squared Error

► Error Squared

Squared Error Loss

► Mean Squared Error

Stacked Generalization

Synonyms

Stacking

Definition

Stacking is an ► ensemble learning technique. A set of models are constructed from bootstrap samples of a dataset, then their outputs on a hold-out dataset are used as *input* to a "meta"-model. The set of base models are called *level*-0, and the meta-model *level*-1. The task of the level-1 model is to combine the set of outputs so as to correctly classify the target, thereby correcting any mistakes made by the level-0 models.

Recommended Reading

Wolpert DH (1992) Stacked generalization. Neural Netw 5(2):241–259

Stacking

► Stacked Generalization

Starting Clause

► Bottom Clause

State

In a ► Markov decision process, *states* represent the possible system configurations facing the decision-maker at each *decision epoch*. They must contain all variable information relevant to the decision-making process.

Statistical Learning

► Inductive Learning

Statistical Machine Translation

Miles Osborne
University of Edinburgh, Edinburgh, UK

Synonyms

SMT

Definition

Statistical machine translation (SMT) deals with automatically mapping sentences in one human language (for example, French) into another human language (such as English). The first language is called the *source* and the second language is called the *target*. This process can be thought of as a stochastic process. There are many SMT variants, depending upon how translation is modeled. Some approaches are in terms of a string-to-string mapping, some use trees-to-strings, and some use tree-to-tree models. All share in common the central idea that translation is automatic, with models estimated from parallel corpora (source-target pairs) and also from monolingual corpora (examples of target sentences).

Motivation and Background

Machine Translation has widespread commercial, military, and political applications. For example, increasingly, the Web is accessed by non-English speakers reading non-English pages. The ability to find relevant information clearly should not be bounded by our language-speaking capabilities. Furthermore, we may not have sufficient linguists in some language of interest to cope with the sheer volume of documents that we would like translated. Enter automatic translation. Machine translation poses a number of interesting machine learning challenges: data sets are typically very large, as are the associated models; the training material used is often noisy and plagued with sparse statistics; the search space of possible translations is sufficiently large that exhaustive search is not possible. Advances in machine learning, such as maximum-margin methods, frequently appear in translation research. SMT systems are now sufficiently mature that they can be deployed in production systems. A good example of this is Google's online Arabic-English translation, which is based upon SMT techniques.

Structure of the Learning System

Modeling

Formally, translation can be described as finding the most likely target sentence e^* for some source sentence f:

$$e^* = \operatorname{argmax}_e P(f|e)P(e)$$

(e conventionally stands for English and f for French, but any language pairs can be substituted.)

This approach has three major aspects:

- A translation model ($P(f|e)$), which specifies the set of possible translations for some target sentence. The translation model also assigns probabilities to these translations, representing their relative correctness.
- A language model ($P(e)$), which models the fluency of the proposed target sentence. This assigns a distribution over strings, with higher probabilities being assigned to sentences which are more representative of natural language. Language models are usually smoothed n-gram models, typically conditioning on two (or more) previous words when predicting the probability of the current word.
- A search process (the argmax operation), which is concerned with navigating through the space of possible target translations. This is called *decoding*. Decoding for SMT is NP-hard, so most approaches use a beam search.

This is called the *Source-Channel* approach to translation (Brown et al. 1994). Most modern SMT systems instead use a ▸ log-linear model, as it is more flexible and allows for various aspects of translation to be balanced together (Och and Ney 2001):

$$e^* = \operatorname{argmax}_e \left(\sum_i fi(e, f)\lambda_i \right)$$

Here, feature functions $f_i(e, f)$ capture some aspect of translation and each feature function has an associated weight λ_i. When we have the two feature functions $P(f|e)$ and $P(e)$, we have the Source-Channel model. The weights are scaling factors (balancing the contributions that each feature function makes) and are optimized with respect to some ▸ loss function which evaluates translation quality. Frequently, this is in terms of the *BLEU* evaluation metric Papineni et al. (2001). Typically, the error surface is nonconvex and the loss function is nondifferentiable, so search techniques which do not use first-order derivatives must be employed. It is worth noting that machine translation evaluation is a complex problem and that methods such as BLEU are not without criticism.

SMT systems usually decompose entire sentences into a sequence of strings called *phrases* (Koehn et al. 2003). The modeling task then becomes one of determining how to break a source sentence into a sequence of contiguous phrases and how to specify which source phrase should be associated with each target phrase. Figure 1 shows an example English-French sentence pair. Figure 2 shows that sentence pair decomposed into phrase-pairs. Phrase-based systems represented an advance over previous word-based models, since phrase-based translation can capture local (within a phrase) word order. Furthermore, phrase-based translation approaches need to make fewer decisions than word-based models. This means there are fewer errors to make.

Those people have grown up, lived and worked for many years in a farming district.
Ces gens ont grandi, vécu et oeuvré des dizaines d'années dans le domain agricole.

Statistical Machine Translation, Fig. 1 A sentence pair

Ces gens ont	Those people have
gens ont grandi	people have grown up
ont grandi ,	have grown up ,
grandi , vécu	grown up , lived
, vécu et	, lived and
vécu et oeuvré	lived and worked
et oeuvré des dizaines d' oeuvré	and worked many
oeuvré des dizaines d' années dizaines	worked many years
des dizaines d' années dans	many years in
années dans le	years in a
le domaine agricole	a farming districtle
domaine agricole .	farming district .

Statistical Machine Translation, Fig. 2 Example phrase pairs

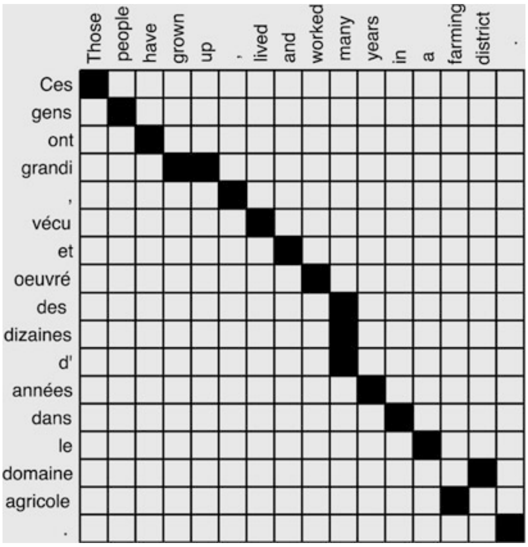
A major aspect of any SMT approach is dealing with phrasal *reordering*. Typically, the translation of each source phrase need not follow the same temporal order in the target sentence. Simple approaches model the absolute distance a target phrase can "move" from the originating target phrase. More sophisticated reordering models condition this movement upon the aspects of the phrase pair.

Our description of SMT is in terms of a string-to-string model. There are numerous other SMT approaches, for example those which use notions of syntax (Chiang 2005). These models are now showing promising results, but are significantly more complex to describe.

Estimation

The translation model of a SMT system is estimated using *parallel corpora*. Because the search space is so large and that parallel corpora is not aligned at the word level, the estimation process is based upon a large-scale application of Expectation-Maximization, along with heuristics. This consists of the following steps:

- Determine how each source word translates to zero or more target words. The IBM models are used for this task, which are based upon the Expectation-Maximization algorithm for parameter estimation (Brown et al. 1994).
- Repeat this process, but instead determine how each target word translates to zero or more source words.
- Harmonize the previous two steps, creating a set of *word alignments* for each sentence pair. This process is designed to use the two directions as alternative views on how words should be translated. Figure 3 shows the sentence pair aligned at the word level.
- Heuristically, determine which sequence of source words translates to a sequence of target words. This produces a set of *phrase-pairs*: a snippet of text in the source sentence and



	Those	people	have	grown	up	,	lived	and	worked	many	years	in	a	farming	district	.
Ces	■															
gens		■														
ont			■													
grandi				■	■											
,						■										
vécu							■									
et								■								
oeuvré									■							
des										■						
dizaines										■						
d'										■						
années											■					
dans												■				
le													■			
domaine															■	
agricole														■		
.																■

Statistical Machine Translation, Fig. 3 The sentence pair in Fig. 1 aligned at the word-level

the associated snippet of text in the target sentence.

- Relative frequency estimators can then be used to characterize how each source phrase translates to a given target phrase.

Parallel corpora varies in size tremendously; for language pairs such as Arabic to English, we have on the order of ten million sentence pairs. Most other language pairs (for example, Finnish to Irish) will have far smaller parallel corpora available. Parallel corpora exists for all European languages and for many other pairs, such as Mandarin to English.

The language model is instead estimated from monolingual corpora, typically using relative frequency estimates, which are then smoothed. For languages such as English, typically billion (and more) words are used. Deploying such large models can pose significant engineering challenges. This is because the language model can easily be so large that it will not fit into the memory of conventional machines. Also, the language model can be queried millions of times when translating sentences, which precludes storing it on disk.

Programs and Data

All of the code and data necessary to begin work on SMT is available either as public source, or for a small payment (in the case of corpora from the LDC):

- The standard software to estimate word-based translation models is Giza++: http://www.fjoch.com/GIZA++.html
- Converting word-based to phrase-based models and decoding can be achieved using the Moses decoder and associated sets of scripts: http://www.statmt.org/jhuws/?n=Moses.HomePage
- Translation performance can be evaluated using BLEU: http://www.nist.gov/speech/tests/mt/resources/scoring.htm
- The SRILM is the standard toolkit for building and using language models: http://www.speech.sri.com/projects/srilm/
- Europarl is a set of parallel corpora, dealing with European languages: http://www.statmt.org/europarl/
- The Linguistics Data Consortium (LDC) maintains corpora of various kinds, including large volumes of monolingual data which can be used to train language models: http://www.ldc.upenn.edu/

Recommended Reading

Brown PF, Pietra SD, Pietra VJD, Mercer RL (1994) The mathematic of statistical machine translation: parameter estimation. Comput Linguist 19(2):263–311

Chiang D (2005) A hierarchical phrase-based model for statistical machine translation. In: Proceedings of the 43rd annual meeting of the association for computational linguistics (ACL'05). Association for Computational Linguistics, Ann Arbor, pp 263–270

Koehn P, Och FJ, Marcu D (2003) Statistical phrase-based translation. In: NAACL '03: proceedings of the 2003 conference of the north american chapter of the association for computational linguistics on human language technology. Association for Computational Linguistics, Morristown, pp 48–54

Och FJ, Ney H (2001) Discriminative training and maximum entropy models for statistical machine translation. In: ACL '02: proceedings of the 40th annual meeting on association for computational linguistics. Association for Computational Linguistics, Morristown, pp 295–302

Papineni K, Roukos S, Ward T, Zhu W-J (2001) Bleu: a method for automatic evaluation of machine translation. In: ACL '02: proceedings of the 40th annual meeting on association for computational linguistics. Association for Computational Linguistics, Morristown, pp 311–318

Statistical Natural Language Processing

▶ Maximum Entropy Models for Natural Language Processing

Statistical Physics of Learning

▶ Phase Transitions in Machine Learning

Statistical Relational Learning

Luc De Raedt[1] and Kristian Kersting[2,3]
[1]Department of Computer Science, Katholieke Universiteit Leuven, Heverlee, Leuven, Belgium
[2]Knowledge Discovery, Fraunhofer IAIS, Sankt Augustin, Germany
[3]Technische Universität Dortmund, Dortmund, Germany

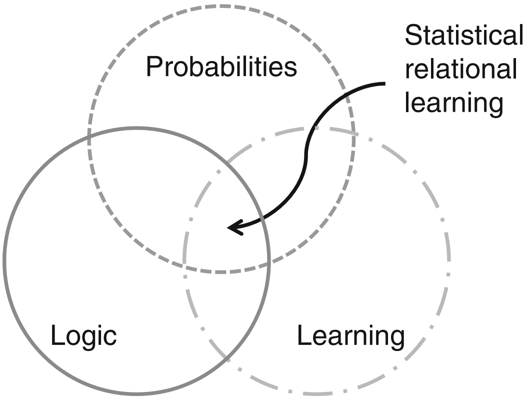
Definition

Statistical relational learning a.k.a. probabilistic inductive logic programming deals with machine learning and data mining in relational domains where observations may be missing, partially observed, or noisy. In doing so, it addresses one of the central questions of artificial intelligence – the integration of probabilistic reasoning with machine learning and first-order and relational representations – and deals with all related aspects such as reasoning, parameter estimation, and structure learning.

Motivation and Background

One of the central questions of artificial intelligence is concerned with combining expressive knowledge representation formalisms such as relational and first-order logic with principled probabilistic and statistical approaches to inference and learning. While traditionally relational and logical representations, probabilistic and statistical reasoning, and machine learning have been studied independently of one another, statistical relational learning investigates them jointly, cf. Fig. 1. A major driving force is the explosive growth in the amount of heterogeneous data that is being collected in the business and scientific world in domains such as bioinformatics, transportation systems, communication networks, social network analysis, citation analysis, and robotics. Characteristic for these domains is that they provide *uncertain* information about varying numbers of entities and relationships among the entities, that is, about *relational* domains. Traditional machine learning approaches are able to cope either with uncertainty or with relational representations but typically not with both.

Many formalisms and representations have been developed in statistical relational learning. For instance, Eisele (1994) has introduced a probabilistic variant of comprehensive unification for-



Statistical Relational Learning, Fig. 1 Statistical relational learning a.k.a. probabilistic inductive logic programming combines probability, logic, and learning malism (CUF). In a similar manner, Muggleton (1996) and Cussens (1999) have upgraded stochastic grammars toward *stochastic logic programs*. Sato (1995) has introduced *probabilistic distributional semantics* for logic programs. Taskar et al. (2002) have upgraded Markov networks toward *relational Markov networks*, and Richardson and Domingos (2006) toward *Markov logic networks*. Neville and Jensen (2004) have extended dependency networks toward *relational dependency networks*. Another research stream has investigated logical and relational extensions of Bayesian networks. It includes Poole's *independent choice logic* (Poole 1993), Ngo and Haddawy's *probabilistic logic programs* (Ngo and Haddawy 1997), Jäger's *relational Bayesian networks* (Jäger 1997), Koller, Getoor, and Pfeffer's *probabilistic relational models* (Getoor 2001; Pfeffer 2000), and Kersting and De Raedt's *Bayesian logic programs* (Kersting and De Raedt 2007).

The benefits of employing logical abstraction and relations within statistical learning are manyfold:

1. Relations among entities allow one to use information about one entity to help reach conclusions about other, related entities.
2. Variables, that is, placeholders for entities allow one to make abstraction of specific entities.
3. Unification allows one to share information among entities. Thus, instead of learning regularities for each single entity independently, statistical relational learning aims at finding general regularities among groups of entities.
4. The learned knowledge is often declarative and compact, which makes it easier for people to understand and to validate.
5. In many applications, there is a rich background theory available, which can efficiently and elegantly be represented as a set of general regularities. This is important because background knowledge may improve the quality of learning as it focuses the learning on the relevant patterns, that is, it restricts the search space.
6. When learning a model from data, relational and logical abstraction allow one to reuse experience in that *learning about one entity improves the prediction for other entities*; and this may even generalize to objects that have never been observed before.

Thus, relational and logical abstraction make statistical learning more robust and efficient. This has proven to be beneficial in many fascinating real-world applications in citation analysis, web mining, natural language processing, robotics, bio- and chemo-informatics, electronic games, and activity recognition.

Theory

Whereas most of the existing works on statistical relational learning have started from a statistical and probabilistic learning perspective and extended probabilistic formalisms with relational aspects, statistical relational learning can elegantly be introduced by starting from ▸ inductive logic programming (De Raedt 2008; Muggleton and De Raedt 1994), which is often also called *multi-relational data mining* (MRDM) (Džeroski and Lavrač 2001). Inductive logic programming is a research field at the intersection of machine learning and logic programming. It forms a formal framework and has introduced practical algorithms for inductively learning relational descriptions (in the form of logic programs) from examples and background knowledge. So, the only difference to statistical relational learning is that it does not explicitly deal with uncertainty.

Essentially, there are only two changes to apply to inductive logic programming approaches in order to arrive at statistical relational learning:

1. ▸ clauses (i.e., logical formulae that can be interpreted as rules; cf. below) are annotated with probabilistic information such as conditional probabilities; and
2. the covers relation (which states the conditions under which a hypothesis considers an example as positive) becomes probabilistic.

A probabilistic covers relation softens the hard covers relation employed in traditional inductive logic programming and is defined as the probability of an example given the hypothesis and the background theory.

Definition 1 (Probabilistic Covers Relation) A probabilistic covers relation takes as arguments an example e, a hypothesis H and possibly the background theory B, and returns the probability value $\mathbf{P}(e \mid H, B)$ between 0 and 1 of the example e given H and B, that is, covers$(e, H, B) = \mathbf{P}(e \mid H, B)$.

It specifies the likelihood of the example given the hypothesis and the background theory. Different choices of the probabilistic covers relation lead to different statistical relational learning approaches; this is akin to the learning settings in inductive logic programming.

Statistical Relational Languages

There is a multitude of different languages and formalisms for statistical relational learning. For an overview of these languages we refer to Getoor and Taskar (2007) and De Raedt et al. (2008). Here, we choose two formalisms that are representatives of the two main streams in statistical relational learning. First, we discuss Markov logic (Richardson and Domingos 2006), which upgrades Markov network toward first-order logic, and second, we discuss ProbLog (De Raedt et al. 2007), which is a probabilistic Prolog based on Sato's distribution semantics (Sato 1995). While Markov logic is a typical example of knowledge-based model construction, ProbLog is a probabilistic programming language.

Case Study: Markov Logic Networks

Markov logic combines first-order logic with ▶ Markov networks. The idea is to view logical formulae as soft constraints on the set of possible worlds, that is, on the interpretations (an interpretation is a set of facts). If an interpretation does not satisfy a logical formula, it becomes less probable, but not necessarily impossible as in traditional logic. Hence, the more formulae an interpretation satisfies, the more likely it becomes. In a Markov logic network, this is realized by associating a weight to each formula that reflects how strong the constraint is. More precisely, a Markov logic network consists of a set of weighted clauses $H = \{c_1, \ldots, c_m\}$. (Markov logic networks, in principle, also allow one to use arbitrary logical formulae, not just clauses. However, for reasons of simplicity, we only employ clauses and make some further simplifications.) The weights w_i of the clauses then specify the strength of the clausal constraint.

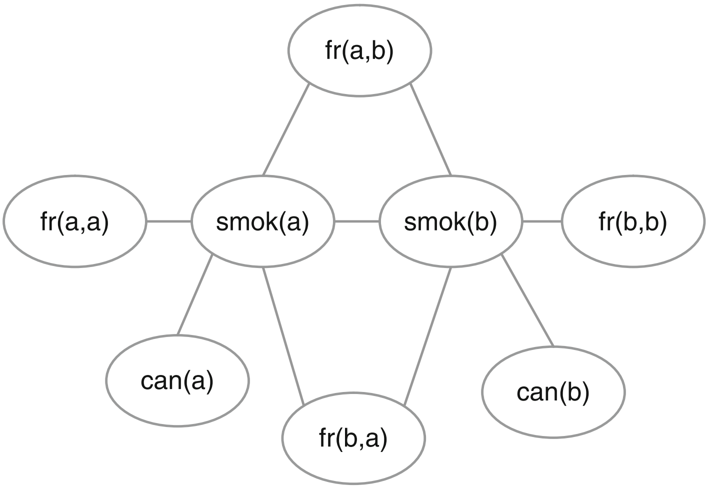
Example 1 Consider the following example (Adapted from Richardson and Domingos 2006). Friends & Smokers is a small Markov logic network that computes the probability of a person having lung cancer on the basis of her friends smoking. This can be encoded using the following weighted clauses:

1.5: cancer(P) ← smoking(P)
1.1: smoking(X) ← friends(X,Y), smoking(Y)
1.1: smoking(Y) ← friends(X,Y), smoking(X)

The first clause states the soft constraint that smoking causes cancer. So, interpretations in which persons that smoke have cancer are more likely than those where they do not (under the assumptions that other properties remain constant). The second and third clauses state that friends of smokers are typically also smokers.

A Markov logic network together with a Herbrand domain (in the form of a set of constants $\{d_1, \ldots, d_k\}$) then induces a grounded Markov network, which defines a probability distribution over the possible Herbrand interpretations.

The nodes, that is, the random variables in the grounded network, are the atoms in the Herbrand base, that is, the facts of the form $p(d'_1, \ldots, d'_n$ where p is a predicate or relation and the d'_i are constants. Furthermore, for every ground instance $c_i\theta$ of a clause c_i in H, there will be an edge between any pair of atoms $a\theta$, $b\theta$ that occurs in $c_i\theta$. The Markov network obtained for the constants anna and bob is shown in Fig. 2. To obtain a probability distribution over the Herbrand interpretations, we still need to define the potentials.



Statistical Relational Learning, Fig. 2 The Markov network for the constants ann and bob (Adapted from Richardson and Domingos 2006)

The probability distribution over interpretations I is

$$\mathbf{P}(I) = \frac{1}{Z} \prod_{c:clause} \mathbf{f}_c(I) \qquad (1)$$

where the f_c are defined as

$$f_c(I) = e^{n_c(I)w_c} \qquad (2)$$

and $n_c(I)$ denotes the number of substitutions θ for which $c\theta$ is satisfied by I, and Z is a normalization constant. The definition of a potential as an exponential function of a weighted feature of a clique is common in Markov networks; cf. ▶ graphical models. The reason is that the resulting probability distribution is easier to manipulate.

Note that for different (Herbrand) domains, different Markov networks will be produced. Therefore, one can view Markov logic networks as a kind of template for generating Markov networks, and, hence, Markov logic is based on knowledge-based model construction. Notice also that Markov logic networks define a probability distribution over interpretations, and nicely separate the qualitative from the quantitative component.

Case Study: ProbLog

Many formalisms do not explicitly encode a set of conditional independency assumptions, as in Bayesian or Markov networks, but rather extend a (logic) programming language with probabilistic choices. Stochastic logic programs (Cussens 2001; Muggleton 1996) directly upgrade stochastic context-free grammars toward definite clause logic, whereas Prism (Sato 1995), probabilistic Horn abduction (PHA) (Poole 1993), and the more recent independent choice logic (ICL) (Poole 1997) specify probabilities on facts from which further knowledge can be deduced. As a simple representative of this stream of work, we introduce the probabilistic Prolog called ProbLog (De Raedt et al. 2007).

The key idea underlying Problog is that some facts f for *probabilistic* predicates are annotated with a probability value. This value indicates the degree of belief, that is the probability, that any ground instance $f\theta$ of f is true. It is also assumed that the $f\theta$ are marginally independent. The probabilistic facts are then augmented with a set of definite clauses defining further predicates (which should be disjoint from the probabilistic ones). An example adapted from De Raedt et al. (2007) is given below.

Example 2 Consider the facts

```
0.9: edge(a,c) ←
0.7: edge(c,b) ←
0.6: edge(d,c) ←
0.9: edge(d,b) ←
```

which specify that with probability 0.9 there is an edge from *a* to *c*. Consider also the following (simplified) definition of *path/2*.

path(X,Y)edge(X,Y) ←
path(X,Y)edge(X,Z), path(Z,Y) ←

One can now define a probability distribution on (ground) proofs as follows. The probability of a ground proof is the product of the probabilities of the (ground) clauses (here, facts) used in the proof. For instance, the only proof for the goal ← path(a,b) employs the facts edge(a,c) and edge(c,b); these facts are marginally independent, and hence the probability of the proof is 0.9 × 0.7. The probabilistic facts used in a single proof are sometimes called an *explanation*.

It is now tempting to define the probability of a ground atom as the sum of the probabilities of the proofs for that atom. However, this does not work without additional restrictions, as shown in the following example.

Example 3 The fact path(d,b) has two explanations:

1. {edge(d,c), edge(c,b)} with probability $0.6 \times 0.7 = 0.42$, and
2. {edge(d,b)} with probability 0.9.

Summing the probabilities of these explanations gives a value of 1.32, which is clearly impossible.

The reason for this problem is that the different explanations are not mutually exclusive, and therefore their probabilities may not be summed. The probability $P(\text{path(d,b)} = \textit{true})$ is, however, equal to the probability that *a* proof succeeds, that is,

$$p(\text{path(d,b)} = true) = P[(\text{e(d,c)} \wedge \text{e(c,b)}) \vee \text{e(d,b)}]$$

which shows that computing the probability of a derived ground fact reduces to computing the probability of a boolean formula in disjunctive normal form (DNF), where all random variables are marginally independent of one another. Computing the probability of such formulae is an NP-hard problem, the *disjoint-sum* problem. Using the *inclusion-exclusion* principle from set theory, one can compute the probability as

$$\begin{aligned} p(\text{path(d,b)} = true) &= P[(\text{e(d,c)} \wedge \text{e(c,b)}) \\ &\quad \vee \text{e(d,b)}] \\ &= P(\text{e(d,c)} \wedge \text{e(c,b)}) \\ &+ P(\text{e(d,b)}) \\ &- P((\text{e(d,c)} \wedge \text{e(c,b)}) \\ &\quad \wedge \text{e(d,b)}) \\ &= 0.6 \times 0.7 + 0.9 - 0.6 \\ &\quad \times 0.7 \times 0.9 = 0.942 \end{aligned}$$

There exist more effective ways to compute the probability of such DNF formulae (De Raedt et al. 2007), where binary decision diagrams are employed to represent the DNF formulae.

The above example shows how the probability of a specific fact is defined and can be computed. The distribution at the level of individual facts (or goals) can easily be generalized to a possible world semantics, specifying a probability distribution on interpretations. It is formalized in the *distribution semantics* of Sato (1995), which is defined by starting from the set of all probabilistic ground facts F for the given program. For simplicity, we shall assume that this set is finite, though Sato's results also hold for the infinite case. The distribution semantics then starts from a probability distribution $P_F(S)$ defined on subsets $S \subseteq F$:

$$P_F(S) = \prod_{f \in s} P(f) \prod_{f \notin s} (1 - P(f)). \quad (3)$$

Each subset S is now interpreted as a set of logical facts and combined with the definite clause program R that specifies the logical part of the probabilistic logic program. Any such combination $S \cup R$ possesses a unique least Herbrand model $M(S \cup R)$, which corresponds to a possible world. The probability of such a possible world is then the sum of the probabilities of the subsets S yielding that possible world, that is,

$$P_W(M) = \sum_{s \subseteq F: M(S \cup R) = M} P_F(S) \quad (4)$$

For instance, in the path example, there are 16 possible worlds, which can be obtained from the 16 different truth assignments to the facts, and whose probabilities can be computed using Eq. (4). As for graphical models, the probability of any logical formula can be computed from a possible world semantics (specified here by P_W).

Because computing the probability of a fact or goal under the distribution semantics is hard, systems such as Prism (Sato 1995) and PHA (Poole 1993) impose additional restrictions that can be used to improve the efficiency of the inference procedure. The key assumption is that the explanations for a goal are *mutually exclusive*, which overcomes the disjoint-sum problem. If the different explanations of a goal do not overlap, then its probability is simply the sum of the probabilities of its explanations. This directly follows from the inclusion-exclusion formulae as under the exclusive-explanation assumption the conjunctions (or intersections) are empty.

Learning

Essentially, any statistical relational approach can be viewed as lifting a traditional inductive logic programming setting by associating probabilistic information to clauses and by replacing the deterministic coverage relation by a probabilistic one. In contrast to traditional graphical models such as Bayesian networks or Markov networks, however, we can also employ "counterexamples" for learning. Consider a simple kinship domain. Assume rex is a male person. Consequently, he cannot be the daughter of any other person, say ann. Thus, daughter(rex,ann) can be listed as a negative example although we will never observe it. "Counterexamples" conflict with the usual view on learning examples in statistical learning.

In statistical learning, we seek to find that hypothesis H^*, which is most likely given the learning examples:

$$\begin{aligned} H^* &= \arg\max_H P(H|E) \\ &= \arg\max_H \frac{P(E|H) \cdot P(F)}{P(E)} \\ &\quad \text{with } P(E) > 0. \end{aligned}$$

Thus, examples E in traditional statistical learning are always observable, that is, $P(E) > 0$. However, in statistical relational learning, as in inductive logic programming, we may also employ "counterexamples" such as daughter(rex,ann), which have probability "0," and that actually never can be observed.

Definition 2 (SRL Problem) **Given** a set $E = E_p \cup E_i$ of *positive* and *negative* examples E_p and E_i (with $E_p \cap E_i = \emptyset$) over some example language $\mathcal{L}_E$, a probabilistic covers relation $covers(e, H, B) = P(e \mid H, B)$, a probabilistic logical language $\mathcal{L}_H$ for hypotheses, and a background theory B, **find** a hypothesis H^* in $\mathcal{L}_H$ such that $H^* = \text{argmax}_H\ score(E, H, B)$ and the following constraints hold: $\forall\ e_p \in E_p\ :\ covers(e_p, H^*, B) > 0$ and $\forall\ e_i \in E_i\ :\ covers(ei, H^*, B) = 0$. The score is some objective function, usually involving the probabilistic covers relation of the observed examples such as the observed likelihood $\prod_{e_p \in E_p} covers(e_p, H^*, B)$ or some penalized variant thereof.

This learning setting unifies inductive logic programming and statistical learning in the following sense: using a deterministic covers relation (either 1 or 0), it yields the classical inductive logic programming learning problem; sticking to propositional logic and learning from *positive* examples, that is, $P(E) > 0$, only yields traditional statistical learning.

To come up with algorithms solving the SRL problem, say for density estimation, one typically distinguishes two subtasks because $H = (L, \lambda)$ is essentially a logical theory L annotated with probabilistic parameters λ:

1. *Parameter estimation* where it is assumed that the underlying logic program L is fixed, and

the learning task consists of estimating the parameters λ that maximize the likelihood.

2. *Structure learning* where both L and λ have to be learned from the data.

In the following paragraphs, we will sketch the basic parameter estimation and structure learning techniques, and illustrate them for each setting.

Parameter Estimation

The problem of parameter estimation is concerned with estimating the values of the parameters λ of a fixed probabilistic program $H = (L, \lambda)$ that best explains the examples E. So, λ is a set of parameters and can be represented as a vector. As already indicated above, to measure the extent to which a model fits the data, one usually employs the likelihood of the data, that is, $P(E|L, \lambda)$, though other scores or variants could be used as well.

When all examples are fully observable, maximum likelihood reduces to frequency counting. In the presence of missing data, however, the maximum likelihood estimate typically cannot be written in closed form. It is a numerical optimization problem, and all known algorithms involve nonlinear optimization. The most commonly adopted technique for probabilistic logic learning is the expectation-maximization (EM) algorithm (Dempster et al. 1977; McLachlan and Krishnan 1997). EM is based on the observation that learning would be easy (i.e., correspond to frequency counting), if the values of all the random variables would be known. Therefore, it estimates these values, maximizes the likelihood based on the estimates, and then iterates. More specifically, EM assumes that the parameters have been initialized (e.g., at random) and then iteratively performs the following two steps until convergence:

(E-Step) On the basis of the observed data and the present parameters of the model, it computes a distribution over all possible completions of each partially observed data case.

(M-Step) Treating each completion as a fully observed data case weighted by its probability, it computes the improved parameter values using (weighted) frequency counting.

The frequencies over the completions are called the *expected counts*. Examples for parameter estimation of probabilistic relational models can be found in Getoor and Taskar (2007) and De Raedt et al. (2008).

Structure Learning

The problem is now to learn both the structure L and the parameters λ of the probabilistic program $H = (L, \lambda)$ from data. Often, further information is given as well. As in inductive logic programming, the additional knowledge can take various different forms, including a ▸ *language bias* that imposes restrictions on the syntax of L, and an *initial hypothesis* (L, λ) from which the learning process can start.

Nearly all (score-based) approaches to structure learning perform a heuristic search through the space of possible hypotheses. Typically, hill-climbing or beam-search is applied until the hypothesis satisfies the logical constraints and the *score*(H, E) is no longer improving. The steps in the search-space are typically made using refinement operators, which make small, syntactic modifications to the (underlying) logic program.

At this point, it is interesting to observe that the logical constraints often require that the positive examples are covered in the logical sense. For instance, when learning ProbLog programs from entailment, the observed example clauses must be entailed by the logic program. Thus, for a probabilistic program $H = (L_H, \lambda_H)$ and a background theory $B = (L_B, \lambda_B)$ it holds that $\forall e_p \in E_p: P(e|H, B) > 0$ if and only if *covers*$(e, L_H, L_B) = 1$, where L_H (respectively L_B) is the underlying logic program (logical background theory) and *covers*(e, L_H, L_B) is the purely logical *covers* relation, which is either 0 or 1.

Applications

Applications of statistical relational learning can be found in many areas such as web search and mining, text mining, bioinformatics, natural language processing, robotics, and social network analysis, among others. Due to space restrictions, we will only name a few of these exciting applications.

For instance, Getoor et al. (2001) have used statistical relational models to estimate the result size of complex database queries. Segal et al. have employed probabilistic relational models to cluster gene expression data (Segal et al. 2001) and to discover cellular processes from gene expression data (Segal et al. 2003). Getoor et al. have used probabilistic relational models to understand tuberculosis epidemiology (Getoor et al. 2004). McGovern et al. (2003) have estimated probabilistic relational trees to discover publication patterns in high-energy physics. Probabilistic relational trees have also been used to learn to rank brokers with respect to the probability that they would commit a serious violation of securities regulations in the near future (Neville et al. 2005). Anguelov et al. (2005) have used relational Markov networks for segmentation of 3D scan data. Markov networks have also been used to compactly represent object maps and to estimate trajectories of people (Limketkai et al. 2005). Kersting et al. have employed relational hidden Markov models for protein fold recognition (Kersting et al. 2006). Poon and Domingos (2008) have shown how to use Markov logic to perform joint unsupervised coreference resolution. Xu et al. have used nonparametric relational models for analyzing social networks (Xu et al. 2010). Kersting and Xu (2009) have used relational Gaussian processes for learning to rank search results. Recently, Poon and Domingos (2009) have shown how to perform unsupervised semantic parsing using Markov logic networks.

Future Directions

We have provided an overview of the new and exciting area of statistical relational learning. It combines principles of probabilistic reasoning, logical representation, and statistical learning into a coherent whole. The techniques of probabilistic logic learning were analyzed starting from an inductive logic programming perspective by lifting the coverage relation to a probabilistic one and annotating the logical formulae. Different choices of the probabilistic coverage relation lead to different representational formalisms, two of which were introduced.

Statistical relational learning is an active area of research within the machine learning and the artificial intelligence community. First, there is the issue of *efficient inference* and learning. Most current inference algorithms for statistical relational models require explicit state enumeration, which is often impractical: the number of states grows very quickly with the number of domain objects and relations. *Lifted* inference algorithms seek to avoid explicit state enumeration and directly work at the level of groups of atoms, eliminating all the instantiations of a set of atoms in a single step, in some cases independently of the number of these instantiations. Despite various approaches to lifted inference (de Salvo Braz et al. 2005; Jaimovich et al. 2007; Kersting et al. 2009; Kisynski and Poole 2009; Milch et al. 2008; Poole 2003; Sen et al. 2008; Singla and Domingos 2008), it largely remains a challenging problem. For what concerns learning, advanced principles of both statistical learning and logical and relational learning can be employed for learning the parameters and the structure of probabilistic logics such as statistical *predicate invention* (Kok and Domingos 2007) and *boosting* (Gutmann and Kersting 2006). Recently, people started to investigate *learning from weighted examples* (see e.g., Chen et al. 2008) and to link statistical relational learning to support vector machines (see e.g., Passerini et al. 2006). Second, there is the issue of *closed-world versus open-world* assumption that is, do we know how many objects there are (see e.g., Milch et al. 2005). Third, there is interest in dealing with *continuous values* within statistical relational learning (see e.g., Chu et al. 2006; Silva et al. 2007; Wang and Domingos 2008; Xu et al. 2009). This is mainly motivated by the fact that most real-world applications actually contain continuous values. *Nonparametric Bayesian* approaches to statistical relational learning have also been developed (see e.g., Kemp et al. 2006; Xu et al. 2006; Yu and

Chu 2007; Yu et al. 2006), to overcome the typically strong parametric assumptions underlying current statistical relational learning. People have also started to investigate *relational variants of classical statistical learning tasks* such as matrix factorizations (see e.g., Singh and Gordon 2008). Finally, while statistical relational learning approaches have been used successfully in a number of applications, they do not yet cope with the *dynamic environments* in an effective way.

Cross-References

- ▸ Multi-relational Data Mining
- ▸ Relational Learning

Recommended Reading

In addition to the references embedded in the text above, we also recommend De Raedt et al. (2008), Getoor and Taskar (2007), De Raedt (2008) and the SRL tutorials at major artificial intelligence and machine learning conferences.

Anguelov D, Taskar B, Chatalbashev V, Koller D, Gupta D, Heitz G et al (2005) Discriminative learning of Markov random fields for segmentation of 3D scan data. In: Schmid C, Soatto S, Tomasi C (eds) IEEE computer society international conference on computer vision and pattern recognition (CVPR-05), San Diego, vol 2, pp 169–176

Chen J, Muggleton S, Santos J (2008) Learning probabilistic logic models from probabilistic examples. Mach Learn 73(1):55–85

Chu W, Sindhwani V, Ghahramani Z, Keerthi S (2006) Relational learning with Gaussian processes. In: Advances in neural information processing systems19 (NIPS-2006). MIT Press, Cambridge

Cussens J (1999) Loglinear models for first-order probabilistic reasoning. In: Blackmond Laskey K, Prade H (eds) Proceedings of the fifteenth annual conference on uncertainty in artificial intelligence (UAI-99), Stockholm. Morgan Kaufmann, San Francisco, pp 126–133

Cussens J (2001) Parameter estimation in stochastic logic programs. Mach Learn J 44(3):245–271

De Raedt L (2008) Logical and relational learning. Springer, Berlin

De Raedt L, Kimmig A, Toivonen H (2007) Problog: a probabilistic Prolog and its application in link discovery. In: Veloso M (ed) Proceedings of the 20th international joint conference on artificial intelligence, Hyderabad, pp 2462–2467

De Raedt L, Frasconi P, Kersting K, Muggleton S (eds) (2008) Probabilistic inductive logic programming. Lecture notes in computer science, vol 4911. Springer, Berlin/Heidelberg

de Salvo Braz R, Amir E, Roth D (2005) Lifted first order probabilistic inference. In: Proceedings of the 19th international joint conference on artificial intelligence (IJCAI-05), Edinburgh, pp 1319–1325

Dempster A, Laird N, Rubin D (1977) Maximum likelihood from incomplete data via the EM algorithm. J R Stat Soc B 39:1–39

Džeroski S, Lavrač N (eds) (2001) Relational data mining. Springer, Berlin

Eisele A (1994) Towards probabilistic extensions of contraint-based grammars. In: Dörne J (ed) Computational aspects of constraint-based linguistics description-II. Institute for Computational Linguistics (IMS-CL), Stuttgart. DYNA-2 deliverable R1.2.B

Getoor L (2001) Learning statistical models from relational data. PhD thesis, Stanford University

Getoor L, Rhee J, Koller D, Small P (2004) Understanding tuberculosis epidemiology using probabilistic relational models. J Artif Intell Med 30:233–256

Getoor L, Taskar B (eds)(2007) Introduction to statistical relational learning. The MIT Press, Cambridge

Getoor L, Taskar B, Koller D (2001) Using probabilistic models for selectivity estimation. In: Proceedings of ACM SIGMOD international conference on management of data, Santa Barbara. ACM Press, pp 461–472

Gutmann B, Kersting K (2006) TildeCRF: conditional random fields for logical sequences. In: Fuernkranz J, Scheffer T, Spiliopoulou M (eds) Proceedings of the 17th European conference on machine learning (ECML-2006), Berlin, pp 174–185

Jäger M (1997) Relational Bayesian networks. In: Laskey K, Prade H (eds) Proceedings of the thirteenth conference on uncertainty in artificial intelligence (UAI-97), Stockholm. Morgan Kaufmann, San Francisco, pp 266–273

Jaimovich A, Meshi O, Friedman N (2007) Template-based inference in symmetric relational Markov random fields. In: Proceedings of the conference on uncertainty in artificial intelligence (UAI-07), Vancouver, pp 191–199

Kemp C, Tenenbaum J, Griffiths T, Yamada T, Ueda N (2006) Learning systems of concepts with an infinite relational model. In: Proceedings of 21st AAAI, Boston

Kersting K, Ahmadi B, Natarajan S (2009) Counting belief propagation. In: Proceedings of the 25th conference on uncertainty in artificial intelligence (UAI-09), Montreal

Kersting K, De Raedt L (2007) Bayesian logic programming: theory and tool. In: Getoor L, Taskar B (eds) An introduction to statistical relational learning. MIT Press, Cambridge, pp 291–321

Kersting K, De Raedt L, Raiko T (2006) Logial Hidden Markov models. J Artif Intell Res (JAIR) 25:425–456

Kersting K, Xu Z (2009) Learning preferences with hidden common cause relations. In: Proceedings of the European conference on machine learning and principles and practice of knowledge discovery in databases (ECML PKDD 09). LNAI. Springer, Bled

Kisynski J, Poole D (2009) Lifted aggregation in directed first-order probabilistic models. In: Boutilier C (ed) Proceedings of the international joint conference on artificial intelligence (IJCAI-09), Pasadena

Kok S, Domingos P (2007) Statistical predicate invention. In: Proceedings of the twenty-fourth international conference on machine learning (ICML-07), Corvallis. ACM Press, pp 433–440

Limketkai B, Liao L, Fox D (2005) Relational object maps for mobile robots. In: Giunchiglia F, Kaelbling LP (eds) Proceedings of the nineteenth international joint conference on artificial intelligence (IJCAI-05), Edinburgh. AAAI Press, pp 1471–1476

McGovern A, Friedland L, Hay M, Gallagher B, Fast A, Neville J et al (2003) Exploiting relational structure to understand publication patterns in high-energy physics. SIGKDD Explor 5(2):165–173

McLachlan G, Krishnan T (1997) The EM algorithm and extensions. Wiley, New York

Milch B, Marthi B, Russell S, Sontag D, Ong D, Kolobov A (2005) BLOG: probabilistic models with unknown objects. In: Giunchiglia F, Kaelbling LP (eds) Proceedings of the nineteenth international joint conference on artificial intelligence (IJCAI-05), Edinburgh. AAAI Press, Edinburgh, pp 1352–1359

Milch B, Zettlemoyer L, Kersting K, Haimes M, Pack Kaelbling L (2008) Lifted probabilistic inference with counting formulas. In: Proceedings of the 23rd AAAI conference on artificial intelligence (AAAI-08), Chicago

Muggleton S (1996) Stochastic logic programs. In: De Raedt L (ed) Advances in inductive logic programming. IOS Press, Amsterdam, pp 254–264

Muggleton S, De Raedt L (1994) Inductive logic programming: theory and methods. J Logic Program 19(20):629–679

Neville J, Jensen D (2004) Dependency networks for relational data. In: Rastogi R, Morik K, Bramer M, Wu X (eds) Proceedings of the fourth IEEE international conference on data mining (ICDM-04), Brighton. IEEE Computer Society Press, pp 170–177

Neville J, Simsek Ö, Jensen D, Komoroske J, Palmer K, Goldberg H (2005) Using relational knowledge discovery to prevent securities fraud. In: Proceedings of the 11th ACM SIGKDD international conference on knowledge discovery and data mining. ACM Press, Chicago

Ngo L, Haddawy P (1997) Answering queries from context-sensitive probabilistic knowledge bases. Theor Comput Sci 171:147–177

Passerini A, Frasconi P, De Raedt L (2006) Kernels on prolog proof trees: statistical learning in the ILP setting. J Mach Learn Res 7:307–342

Pfeffer A (2000) Probabilistic reasoning for complex systems. PhD thesis, Computer Science Department, Stanford University

Poole D (1993) Probabilistic Horn abduction and Bayesian networks. Artif Intell J 64:81–129

Poole D (1997) The independent choice logic for modelling multiple agents under uncertainty. Artif Intell 94(1–2):7–56

Poole D (2003) First-order probabilistic inference. In: Gottlob G, Walsh T (eds) Proceedings of the eighteenth international joint conference on artificial intelligence (IJCAI-03), Acapulco. Morgan Kaufmann, San Francisco, pp 985–991

Poon H, Domingos P (2008) Joint unsupervised coreference resolution with Markov logic. In: Proceedings of the 2008 conference on empirical methods in natural language processing (EMNLP), Honolulu

Poon H, Domingos P (2009) Unsupervised semantic parsing. In: Proceedings of the 2009 conference on empirical methods in natural language processing (EMNLP), Singapore

Richardson M, Domingos P (2006) Markov logic networks. Mach Learn 62:107–136

Sato T (1995) A statistical learning method for logic programs with distribution semantics. In: Sterling L (ed) Proceedings of the twelfth international conference on logic programming (ICLP-95), Tokyo. MIT Press, pp 715–729

Segal E, Battle A, Koller D (2003) Decomposing gene expression into cellular processes. In: Proceedings of Pacific symposium on biocomputing (PSB), Lihue. World Scientific, pp 89–100

Segal E, Taskar B, Gasch A, Friedman N, Koller D (2001) Rich probabilistic models for gene expression. Bioinformatics 17(Suppl 1):S243–252 (Proceedings of ISMB 2001)

Sen P, Deshpande A, Getoor L (2008) Exploiting shared correlations in probabilistic databases. In: Proceedings of the international conference on very large data bases (VLDB-08), Auckland

Silva R, Chu W, Ghahramani Z (2007) Hidden common cause relations in relational learning. In: Advances in neural information processing systems20 (NIPS-2007). MIT Press, Cambridge

Singh A, Gordon G (2008) Relational learning via collective matrix factorization. In: Proceedings of 14th international conference on knowledge discovery and data mining, Las Vegas

Singla P, Domingos P (2008) Lifted first-order belief propagation. In: Proceedings of the 23rd AAAI conference on artificial intelligence (AAAI-08), Chicago, pp 1094–1099

Taskar B, Abbeel P, Koller D (2002) Discriminative probabilistic models for relational data. In: Darwiche A, Friedman N (eds) Proceedings of the eighteenth conference on uncertainty in artificial intelligence (UAI-02), Edmonton, pp 485–492

Wang J, Domingos P (2008) Hybrid markov logic networks. In: Proceedings of the 23rd AAAI conference on artificial intelligence (AAAI-08), Chicago, pp 1106–1111

Xu Z, Kersting K, Tresp V (2009) Multi-relational learning with Gaussian processes. In: Boutilier C (ed) Proceedings of the international joint conference on artificial intelligence (IJCAI-09), Pasadena

Xu Z, Tresp V, Rettinger A, Kersting K (2010) Social network mining with nonparametric relational models. In: Advances in social network mining and analysis. Lecture notes in computer science, vol 5498. Springer, Berlin/Heidelberg

Xu Z, Tresp V, Yu K, Kriegel HP (2006) Infinite hidden relational models. In: Proceedings of 22nd UAI, Cambridge

Yu K, Chu W (2007) Gaussian process models for link analysis and transfer learning. In: Advances in neural information processing systems20 (NIPS-2007). MIT Press, Cambridge

Yu K, Chu W, Yu S, Tresp V, Xu Z (2006) Stochastic relational models for discriminative link prediction. In: Advances in neural information processing systems (NIPS-2006), vol 19. MIT Press, Cambridge

Stochastic Finite Learning

Thomas Zeugmann
Hokkaido University, Sapporo, Japan

Motivation and Background

Assume that we are given a concept class $\mathcal{C}$ and should design a learner for it. Next, suppose we already know or could prove $\mathcal{C}$ not to be learnable in the model of ▶ PAC learning. But it can be shown that $\mathcal{C}$ is learnable within Gold's (1967) model of ▶ inductive inference or learning in the limit. Thus, we can design a learner behaving as follows. When fed any of the data sequences allowed in this model, it converges in the limit to a hypothesis correctly describing the target concept. Nothing more is known. Let M be any fixed learner. If $(d_n)_{n \geq 0}$ is any data sequence, then the *stage of convergence* is the least integer m such that $M(d_m) = M(d_n)$ for all $n \geq m$ provided such an n exists (and infinite, otherwise). In general, it is undecidable whether or not the learner has already reached the stage of convergence, but even if it is decidable for a particular concept class, it may be practically infeasible to do so. This *uncertainty* may not be tolerable in many applications.

When we tried to overcome this uncertainty, the idea of stochastic finite learning emerged. Clearly, in general nothing can be done, since in Gold's (1967) model the learner has to learn from any data sequence. So for every concept that needs more than one datum to converge, one can easily construct a sequence where the first datum is repeated very often and where therefore the learner does not find the right hypothesis within the given bound. However, such data sequences seem unnatural. Therefore, we looked at data sequences that are generated with respect to some probability distribution taken from a prespecified class of probability distributions and computed the expected *total learning time*, i.e., the expected time until the learner reaches the stage of convergence (cf. Erlebach et al. 2001; Zeugmann 1998). Clearly, one is then also interested in knowing how often the expected total learning time is exceeded. In general, Markov's inequality can be applied to obtain the relevant tail bounds. However, if the learner is known to be rearrangement-independent and conservative, then we always get *exponentially* shrinking tail bounds (cf. Rossmanith and Zeugmann 2001). A learner is said to be *rearrangement-independent* if its output depends exclusively on the range and length of its input (but not on the order) (cf., e.g., Lange and Zeugmann (1996) and the references therein). Furthermore, a learner is *conservative*, if it exclusively performs mind changes that can be justified by an inconsistency of the abandoned hypothesis with the data received so far (see Angluin (1980b) for a formal definition).

Combining these ideas results in stochastic finite learning. A stochastic finite learner is successively fed data about the target concept. Note that these data are generated randomly with respect to one of the probability distributions from the class of underlying probability distributions. Additionally, the learner takes a confidence parameter δ as input. But in contrast to learning in the limit,

S

the learner itself decides how many examples it wants to read. Then it computes a hypothesis, outputs it, and stops. The hypothesis output is correct for the target with probability at least $1 - \delta$.

The description given above explains how it works, but not why it does. Intuitively, the stochastic finite learner simulates the limit learner until an upper bound for twice the expected total number of examples needed until convergence has been met. Assuming this to be true, by Markov's inequality the limit learner has now converged with probability $1/2$. All what is left is to decrease the probability of failure. This can be done by using again Markov's inequality, i.e., increasing the sample complexity by a factor of $1/\delta$ results in a confidence of $1 - \delta$ for having reached the stage of convergence.

Note that the stochastic finite learner has to calculate an upper bound for the stage of convergence. This is precisely the point where we need the parameterization of the class $\mathcal{D}$ of underlying probability distributions. Then a bit of *prior knowledge* must be provided in the form of suitable upper and/or lower bounds for the parameters involved. A more serious difficulty is to incorporate the unknown target concept into this estimate. This step depends on the concrete learning problem on hand and requires some extra effort.

It should also be noted that our approach may be beneficial even in case that the considered concept class is PAC learnable.

Definition

Let $\mathcal{D}$ be a set of probability distributions on the learning domain, let $\mathcal{C}$ be a concept class, $\mathcal{H}$ a hypothesis space for $\mathcal{C}$, and let $\delta \in (0, 1)$. The pair $(\mathcal{C}, \mathcal{D})$ is said to be *stochastically finitely learnable with δ-confidence* with respect to $\mathcal{H}$ iff there is a learner M that for every $c \in \mathcal{C}$ and every $D \in \mathcal{D}$ performs as follows. Given any random data sequence θ for c generated according to D, M stops after having seen a finite number of examples and outputs a single hypothesis $h \in \mathcal{H}$. With probability at least $1 - \delta$ (with respect to distribution D), h has to be correct, i.e., $c = h$.

If stochastic finite learning can be achieved with δ-confidence for every $\delta > 0$, then we say that $(\mathcal{C}, \mathcal{D})$ can be learned stochastically finite *with high confidence*.

Detail

Note that there are subtle differences between our model and PAC learning. By its definition, stochastic finite learning is not completely distribution independent. A bit of *additional knowledge* concerning the underlying probability distributions is required. Thus, from that perspective, stochastic finite learning is weaker than the PAC model. On the other hand, we do *not* measure the quality of the hypothesis with respect to the underlying probability distribution. Instead, we require the hypothesis computed to be exactly correct with high probability. Note that exact identification with high confidence has been considered within the PAC paradigm, too (cf., e.g., Goldman et al. 1993). Conversely, we also can easily relax the requirement to learn *probably exactly correct* but whenever possible we shall not do it.

Furthermore, in the uniform PAC model as introduced in Valiant (1984), the sample complexity depends exclusively on the VC dimension of the target concept class and the error and confidence parameters ε and δ, respectively. This model has been generalized by allowing the sample size to depend on the concept complexity, too (cf., e.g., Blumer et al. 1989; Haussler et al. 1991). Provided no upper bound for the concept complexity of the target concept is given, such PAC learners decide themselves how many examples they wish to read (cf. Haussler et al. 1991). This feature is also adopted to our setting of stochastic finite learning. However, all variants of efficient ▶ PAC learning we are aware of require that all hypotheses from the relevant hypothesis space are uniformly polynomially evaluable. Though this requirement may be necessary in some cases to achieve (efficient) stochastic finite learning, it is not necessary in general as we shall see below.

In the following, we provide two sample applications of stochastic finite learning. We always choose as hypothesis space the concept class $\mathcal{C}$ itself.

Learning Monomials

Let $X_n = \{0,1\}^n$ be the learning domain, let $\mathcal{L}_n = \{x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n\}$ (set of literals) and consider the class $\mathcal{C}_n$ of all concepts describable by a conjunction of literals. As usual, we refer to any conjunction of literals as a *monomial*. A monomial m describes a concept $c \subseteq X_n$ in the obvious way: the concept contains exactly those binary vectors for which the monomial evaluates to 1. For a monomial m, let $\#(m)$ denote its length, i.e., the number of literals in it.

The basic ingredient to the stochastic finite learner is Haussler's (1987) Wholist algorithm, and thus the main emphasis is on the resulting complexity. The Wholist algorithm can also be used to achieve ▸PAC learning of the class $\mathcal{C}_n$, and the resulting sample complexity is $O(1/\varepsilon \cdot (n + \ln(1/\delta)))$ for all $\varepsilon, \delta \in (0,1]$. Since the Wholist algorithm learns from positive examples only, it is meaningful to study the learnability of $\mathcal{C}_n$ from positive examples only. So, the stage of convergence is *not* decidable.

Since the Wholist algorithm immediately converges for the empty concept, we exclude it from our considerations. That is, we consider concepts $c \in \mathcal{C}_n$ described by a monomial $m = \bigwedge_{j=1}^{\#(m)} \ell_{i_j}$ such that $k = k(m) = n - \#(m) > 0$. A literal not contained in m is said to be irrelevant. Bit i is said to be irrelevant for monomial m if neither x_i nor $\bar{x}_i$ appears in m. There are 2^k positive examples for c. For the sake of presentation, we assume these examples to be *binomially distributed* with parameter p. So, in a random positive example, all entries corresponding to irrelevant bits are selected independently to one another. With some probability p, this will be a 1, and with probability $1-p$, this will be a 0. Only distributions where $0 < p < 1$ are considered, since otherwise exact identification is impossible. Now, one can show that the expected number of examples needed by the Wholist algorithm until convergence is bounded by $\lceil \log_\psi k(m) \rceil + \tau + 2$, where $\psi := \min\left\{\frac{1}{1-p}, \frac{1}{p}\right\}$ and $\tau := \max\left\{\frac{p}{1-p}, \frac{1-p}{p}\right\}$.

Let *CON* denote a random variable for the stage of convergence. Since the Wholist algorithm is rearrangement-independent and conservative, we can conclude (cf. Rossmanith and Zeugmann 2001)

$$\Pr(\mathrm{CON} > 2t \cdot \mathrm{E}[\mathrm{CON}]) \leq 2^{-t} \quad \text{for all natural numbers } t \geq 1\,. \tag{1}$$

Finally, in order to obtain a stochastic finite learner, we reasonably assume that *prior knowledge* is provided by parameters p_{low} and p_{up} such that $p_{\text{low}} \leq p \leq p_{\text{up}}$ for the true parameter p. Binomial distributions fulfilling this requirement are called $(p_{\text{low}}, p_{\text{up}})$*-admissible distributions*. Let $\mathcal{D}_n[p_{\text{low}}, p_{\text{up}}]$ denote the set of such distributions on X_n. Then one can show *Let* $0 < p_{\text{low}} \leq p_{\text{up}} < 1$ *and* $\psi := \min\{\frac{1}{1-p_{\text{low}}}, \frac{1}{p_{\text{up}}}\}$. *Then* $(\mathcal{C}_n, \mathcal{D}_n[p_{\text{low}}, p_{\text{up}}])$ *is stochastically finitely learnable with high confidence from positive examples. To achieve* δ*-confidence no more than* $O\left(\log_2 1/\delta \cdot \log_\psi n\right)$*, many examples are necessary.*

Therefore, we have achieved an exponential improvement on the number of examples needed for learning (compared to the PAC bound displayed above), and, in addition, our stochastic finite learner exactly identifies the target. Note that this result is due to Reischuk and Zeugmann; however, we refer the reader to Zeugmann (2006) for the relevant proofs.

The results obtained for learnability from positive examples only can be extended *mutatis mutandis* to the case when the learner is fed positive and negative examples (cf. Zeugmann (2006) for details).

Learning Pattern Languages

The pattern languages have been introduced by Angluin (1980a) and can be informally defined as follows. Let $\Sigma = \{0, 1, \ldots\}$ be any finite alphabet containing at least two elements. Let $X = \{x_0, x_1, \ldots\}$ be a countably infinite set of variables such that $\Sigma \cap X = \emptyset$. *Patterns* are nonempty strings over $\Sigma \cup X$, e.g., 01, $0x_0111$, $1x_0x_00x_1x_2x_0$ are patterns. The length of a string $s \in \Sigma^*$ and of a pattern π is denoted by $|s|$ and $|\pi|$, respectively. A pattern π is in *canonical form* provided that if k is the number of different variables in π then the variables occurring in π are precisely $x_0, \ldots, x_{k-1}$. Moreover, for every j with $0 \leq j < k - 1$, the leftmost occurrence of x_j in π is left to the leftmost occurrence of x_{j+1}. The examples given above are patterns in canonical form.

If k is the number of different variables in π, then we refer to π as to a *k-variable pattern*. For example, $x0xx$ is a one-variable pattern, and $x_010x_1x_0$ is a two-variable pattern. If π is a pattern, then the language generated by π is the set of all strings that can be obtained from π by substituting a *nonnull* element $s_i \in \Sigma^*$ for each occurrence of the variable symbol x_i in π, for all $i \geq 0$. We use $L(\pi)$ to denote the language generated by pattern π. So, 1011, 1001010 belong to $L(x0xx)$ (by substituting 1 and 10 for x, respectively) and 010110 is an element of $L(x_010x_1x_0)$ (by substituting 0 for x_0 and 11 for x_1). Note that even the class of all one-variable patterns has infinite ▸ VC dimension (cf. Mitchell et al. 1999).

Reischuk and Zeugmann (2000) designed a stochastic finite learner for the class of all one-variable pattern languages that runs in time $O(|\pi| \log(1/\delta))$ for all meaningful distributions and learns from positive data only. That is, all data fed to the learner belong to the target pattern language. Furthermore, by meaningful distribution essentially the following is meant. The expected length of an example should be finite and the distribution should allow to learn the target pattern. This is then expressed by fixing some suitable parameters. It should be noted that the algorithm is highly practical, and a modification of it also works for the case that *empty* substitutions are allowed. Though this seems to be a minor modification, it is *not*. The learnability results for pattern languages resulting from a definition that also allows for empty substitutions considerably differ from the case, where only nonnull substitutions are admitted (cf. Reidenbach 2006, 2008).

For the class of all pattern languages, one can also provide a stochastic finite learner identifying the whole class from positive data. In order to arrive at a suitable class of distributions, essentially three requirements are made. The first one is the same as in the one-variable case, i.e., the expected length $\mathrm{E}[\Lambda]$ of a generated string should be finite. Second, the class of distributions is restricted to regular product distributions, i.e., for all variables the substitutions are identically distributed.

Third, two parameters α and β are introduced. The parameter α is the probability that a string of length 1 is substituted, and β is the conditional probability that two random strings that get substituted into π are identical under the condition that both have length 1. These two parameters ensure that the target pattern language is learnable at all. The stochastic finite learner is then using as *a priori knowledge* a lower bound α^* for α and an upper bound β^* for β. The basic ingredient to this stochastic finite learner is Lange and Wiehagen's (1991) pattern language learning algorithm. Rossmanith and Zeugmann's (2001) stochastic finite learner for the pattern languages runs in time $O\left((1/\alpha_*^k)\mathrm{E}[\Lambda] \log_{1/\beta_*}(k) \log_2(1/\delta)\right)$, where k is the number of different variables in the target pattern. So, with increasing k it becomes impractical.

Note that the two stochastic finite learners for the pattern languages can compute the expected stage of convergence, since the first string seen provides an upper bound for the length of the target pattern.

For further information, we refer the reader to Zeugmann (2006) and the references therein. More research is needed to explore the potential of stochastic finite learning. Such investigations should extend the learnable classes, should study the incorporation of noise, and should explore further possible classes of meaningful probability distributions.

Cross-References

- ▶ Inductive Inference
- ▶ PAC Learning

Recommended Reading

Angluin D (1980a) Finding patterns common to a set of strings. J Comput Syst Sci 21(1):46–62

Angluin D (1980b) Inductive inference of formal languages from positive data. Inf Control 45(2):117–135

Blumer A, Ehrenfeucht A, Haussler D, Warmuth MK (1989) Learnability and the Vapnik-Chervonenkis dimension. J ACM 36(4):929–965

Erlebach T, Rossmanith P, Stadtherr H, Steger A, Zeugmann T (2001) Learning one-variable pattern languages very efficiently on average, in parallel, and by asking queries. Theor Comput Sci 261(1):119–156

Gold EM (1967) Language identification in the limit. Inf Control 10(5):447–474

Haussler D (1987) Bias, version spaces and Valiant's learning framework. In: Langley P (ed) Proceedings of the fourth international workshop on machine learning. Morgan Kaufmann, San Mateo, pp 324–336

Haussler D, Kearns M, Littlestone N, Warmuth MK (1991) Equivalence of models for polynomial learnability. Inf Comput 95(2):129–161

Lange S, Wiehagen R (1991) Polynomial-time inference of arbitrary pattern languages. New Gener Comput 8(4):361–370

Lange S, Zeugmann T (1996) Set-driven and rearrangement-independent learning of recursive languages. Math Syst Theory 29(6):599–634

Mitchell A, Scheffer T, Sharma A, Stephan F (1999) The VC-dimension of subclasses of pattern languages. In: Watanabe O, Yokomori T (eds) Proceedings of the 10th international conference on algorithmic learning theory, ALT '99, Tokyo, Dec 1999. Lecture notes in artificial intelligence, vol 1720. Springer, pp 93–105

Reidenbach D (2006) A non-learnable class of E-pattern languages. Theor Comput Sci 350(1):91–102

Reidenbach D (2008) Discontinuities in pattern inference. Theor Comput Sci 397(1–3):166–193

Reischuk R, Zeugmann T (2000) An average-case optimal one-variable pattern language learner. J Comput Syst Sci 60(2):302–335

Rossmanith P, Zeugmann T (2001) Stochastic finite learning of the pattern languages. Mach Learn 44(1/2): 67–91

Goldman SA, Kearns MJ, Schapire RE (1993) Exact identification of read-once formulas using fixed points of amplification functions. SIAM J Comput 22(4):705–726

Valiant LG (1984) A theory of the learnable. Commun ACM 27(11):1134–1142

Zeugmann T (1998) Lange and Wiehagen's pattern language learning algorithm: an average-case analysis with respect to its total learning time. Ann Math Artif Intell 23:117–145

Zeugmann T (2006) From learning in the limit to stochastic finite learning. Theor Comput Sci 364(1):77–97. Special issue for ALT 2003

Stopping Criteria

▶ Pre-pruning

Stratified Cross Validation

Stratified Cross Validation is a form of ▶ cross validation in which the class distribution is kept as close as possible to being the same across all folds.

Stream Classification

Jerzy Stefanowski and Dariusz Brzezinski
Institute of Computing Science, Poznan
University of Technology, Poznan, Poland

Abstract

Compared to batch learning from static data, constructing classifiers from data streams implies new requirements for algorithms, such as constraints on memory usage, restricted processing time, and one scan of incoming examples. Additionally, streams classifiers have to adapt to concept drifts. The entry discusses the following stream classification issues: data stream specific requirements, processing schemes, categorization of concept drifts, classifier evaluation criteria and procedures, forgetting mechanisms, change detection methods, main algorithms for

supervised learning of single classifiers and ensembles, open problems, areas of application.

Definition

Stream classification is a variant of incremental learning of classifiers that has to satisfy requirements specific for massive streams of data: restrictive processing time, limited memory, and one scan of incoming examples. Additionally, stream classifiers often have to be adaptive, as they usually act in dynamic, non-stationary environments where data and target concepts can change over time. To fulfill these requirements, new solutions include dedicated data management and forgetting mechanisms, concept drift detectors that monitor the underlying changes in the stream, effective online single classifiers, and adaptive ensembles that continuously react to changes in the stream.

Motivation and Background

In many data-intensive applications, like sensor networks, traffic control, market analysis, Web user tracking, and social media, massive volumes of data are continuously generated in the form of data streams. A data stream is a potentially unbounded, ordered sequence of data items, which arrive continuously at high speeds. These data elements can be simple attribute-value pairs like relational database tuples or more complex structures such as graphs.

The main characteristics of streams include:

- continuous flow (elements arrive one after another),
- huge data volumes (possibly of an infinite length),
- rapid arrival rate (relatively high with respect to processing power of the system),
- susceptibility to change (data distributions generating examples may change on the fly).

Due to the above characteristics, learning from data streams differs from ▶ batch learning, where data are stored in finite, persistent data repositories. The main dissimilarities include the sequential nature of the data, massive volumes, processing speed restrictions, and the fact that data elements cannot be accessed multiple times as it is in the case of learning from static repositories. Moreover, contrary to ▶ online learning, stream classification does not assume adversarial actions from the instance generating process, but rather focuses on computational restrictions.

One of the most widely studied tasks in data stream mining is ▶ supervised classification. Apart from the aforementioned general difficulties connected with learning from streams, classification is also often performed in non-stationary environments, where the data distribution and target concepts can change over time. This phenomenon, called ▶ concept drift, deteriorates the predictive accuracy of classifiers as the instances they were trained on differ from the current data. Typical examples of real-life concept drifts include content changes in unwanted emails in spam categorization or evolving customer preferences.

Several researchers imply the following requirements on algorithms learning classifiers from streams (Bifet et al. 2010):

1. Process one example at a time and inspect it only once.
2. Use a limited amount of memory.
3. Be ready to predict at any time.
4. Be able to react to concept drift in case of evolving data streams.

Typical batch learning algorithms for supervised classification are not capable of fulfilling all of the listed data stream requirements. ▶ Incremental learning is also insufficient, as it does not meet tight computational demands and does not tackle concept drift. Therefore, several new learning algorithms have been introduced. Surveys on stream classification, such as Ditzler et al. (2015), Gama (2010), and Kuncheva (2004), showcase research on using sliding windows to manage memory and provide a forgetting mechanism, sampling techniques, drift detectors, and new online algorithms.

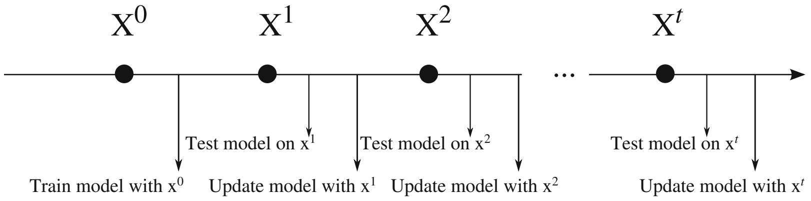
Structure of the Learning System

Stream classification can be formalized as follows. Learning instances from a stream $\mathcal{S}$ appear incrementally as a sequence of labeled ▸ examples $\{\mathbf{x}^t, y^t\}$ for $t = 1, 2, \ldots, T$, where $\mathbf{x}$ is a vector of ▸ attribute values and y is a ▸ class label ($y \in \{K_1, \ldots, K_l\}$). A new example $\mathbf{x}^t$ is classified by a classifier C, which predicts its class label. Here, we consider a completely supervised framework where after some time the true class label y^t is available and can be used to update the classifier.

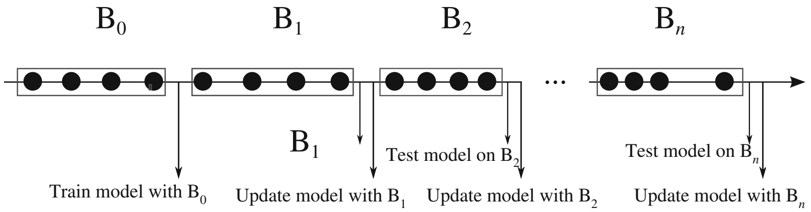
Examples from the data stream can be provided either *online*, i.e., instance by instance, or in portions (*blocks*). In the first approach, presented in Fig. 1, algorithms process single examples appearing one by one in consecutive moments in time, while in the other approach, presented in Fig. 2, examples are available only in larger sets called data blocks (or data chunks) $B_1, B_2, \ldots, B_n$, where n denotes the last element of the stream up to the current timepoint. Blocks are usually of equal size and the construction, evaluation, or updating of classifiers is done when all examples from a new block are available. This distinction also refers to the availability of class labels. For instance, in some problems data elements are naturally accumulated through some time and labeled in blocks. However, with class labels appearing online with single instances, algorithms have the possibility of reacting to concept drift much faster than in block-based environments.

Two basic models of data streams are considered: *stationary*, where examples are drawn from a fixed although unknown probability distribution, and *non-stationary*, where data can evolve over time. As process changes occur in many real-world problems (Zliobaite et al. 2015), most stream classification algorithms are capable of predicting, detecting, and adapting to concept drifts.

Concept drift can be defined from the perspective of hidden data contexts, which are unknown to the learning algorithm. However, in case of evolving streams, a more probabilistic view on the matter can be presented (Gama 2010). In each point in time t, every example is generated by a source with a joint distribution $P^t(\mathbf{x}, y)$ over the data. Concepts in data are *stable* if all examples are generated by the same distribution. If for two distinct points in time t and $t + \Delta$ an $\mathbf{x}$ exists such that $P^t(\mathbf{x}, y) \neq P^{t+\Delta}(\mathbf{x}, y)$, then concept drift



Stream Classification, Fig. 1 Online processing



Stream Classification, Fig. 2 Block processing occurs. Although different component probabilities of $P^t(\mathbf{x}, y)$ may change (Gama et al. 2014), in case of supervised classification, one is mainly interested in *real drift*, i.e., changes in posterior probabilities of classes $P(y|\mathbf{x})$.

Usually two basic types of concept drifts are distinguished: *sudden* (abrupt) and *gradual*. The first type of drift occurs when at a moment in time t, the source data distribution in S^t is suddenly replaced by a different distribution in S^{t+1}. Gradual drifts are not so radical and they are connected with a slower rate of changes that can be noticed while observing a data stream for a longer period of time. In some domains, situations when previous concepts reappear after some time are separately treated and analyzed as *recurring* drifts (Gomes et al. 2014). Moreover, data streams can contain outliers and ▶ noise, but these are not considered as concept drifts and stream classifiers should be robust to these random changes.

Evaluation

Stream classification requirements make *processing time*, *memory usage*, *predictive performance*, and the *ability to adapt* key evaluation criteria.

The time required to process a single instance and the average memory usage should remain constant throughout the life of a stream classifier. That is why training and testing time as well as model size have to be periodically monitored during stream classification. Additionally, processor time and memory are also considered key costs when deploying a stream classification system and are sometimes measured in a single metric called RAM hours.

The predictive performance of stream classifiers is usually assessed using evaluation measures known from static supervised classification, such as ▶ accuracy or ▶ error rate. However, contrary to batch learning scenarios, it is assumed that due to the size and speed of data streams, repeated runs over the data are not necessary to estimate these measures on labeled testing examples. Due to their computational costs, resampling techniques such as ▶ cross-validation or ▶ bootstrapping are deemed too expensive for streams. As a result, simpler error-estimation procedures are used, yet ones that build a picture of performance over time.

One of such evaluation procedures involves using a ▶ holdout test set to periodically evaluate the classifier's performance. An alternate scheme of estimating the performance of stream classifiers involves interleaving testing with training. Each individual example is first used to test the classifier before it is used for training (see Fig. 1). This evaluation procedure, often called *test-then-train*, has the advantage that it makes maximum use of the available data. A similar procedure of interleaving testing with training can also be performed with blocks of examples instead of single instances (see Fig. 2). However, for evolving streams the prequential evaluation procedure is suggested (Gama 2010). The term *prequential* (blend of predictive and sequential) stems from online learning and is used in data stream mining literature to denote algorithms that base their functioning only on the most recent data rather than the entire stream. Such a procedure highlights the current rather than overall performance and, as a result, showcases changes in the stream more clearly, which is especially important for drift detection. All three of the aforementioned evaluation procedures (holdout, test-then-train, prequential) are usually used to periodically calculate a selected metric, e.g., accuracy, and plot its value creating a line chart depicting classifier performance over time.

Finally, an important criterion when comparing stream classifiers is their ability to react to various types of concept changes. Adaptability can be evaluated by comparing drift reaction times. This is done by measuring the time between the start of a drift and the moment when the tested classifier's accuracy recovers to a level from before the drift. More elaborate methods of assessing the classifier's ability to adapt include *recovery analysis* and *controlled permutations* (Krempl et al. 2014). Nevertheless, in order to calculate reaction times and other adaptability measures, usually a human expert needs to determine moments when a drift starts and when a classifier recovers from it. Alternately, such evaluations are carried out with synthetic data generators.

Algorithms

The simplest categorization of algorithms for learning stream classifiers makes a distinction between *single classifiers* and *ensembles*. Additionally, from the perspective of learning from drifting environments, most of researchers distinguish *active* approaches, which trigger changes in classifiers when drifts are detected, and *passive* approaches, which continuously update the classifier regardless of whether drifts occur in the data stream or not (Gama et al. 2014). We discuss algorithms from the point of view of both of these taxonomies.

Data Management and Forgetting Mechanisms

Many approaches to dealing with time-changing streams involve the use of some sort of data management or forgetting mechanism. Data management strategies specify which data is used for learning, while forgetting strategies specify how old data are discarded. Both mechanisms are necessary to meet time and memory requirements posed by data streams and serve as a way of reacting to drifts by eliminating those examples that come from an old concept.

Online classifiers decide if an example will be included in the learning model on a per-instance basis. Such an approach promotes gradual adaptation to evolving concepts mainly by continuously updating the model with new examples. As an alternative, several classifiers apply *sliding windows* to keep the classifier consistent only with the most recent data. As sliding windows encompass a larger set of examples, they can be used to periodically build classifiers by conventional batch algorithms. From this point of view, this data management mechanism can be viewed as a general approach to transforming batch learners into classifiers for concept-drifting data streams.

The basic windowing algorithm is straightforward. Each example updates the window and later the classifier is updated by that window. The key part of this algorithm lies in the definition of the window, i.e., in the way it models the forgetting process. In the simplest approach, sliding windows are of fixed size and include only the most recent examples from the data stream. With each new data point, the oldest example that does not fit in the window is discarded. More complex approaches vary the window size depending on, e.g., the indications of a drift detector (Bifet and Gavaldà 2007).

Sliding windows are also one of the most popular forgetting mechanisms – examples that fall outside of the window are instantly excluded from the model. From this perspective, two basic types of windows are defined: *sequence based*, where the size of a window is characterized by the number of instances, and *timestamp based*, where the size is defined by duration time.

There are two common alternatives to forgetting using sliding windows: *sampling* and *fading factors*. The first alternative aims at summarizing the characteristics of the data stream over a long period of time using a limited number of examples. One of the best known data stream sampling algorithms is reservoir sampling, which keeps a fixed-size sample of the stream that is updated with randomly selected instances (Aggarwal 2007). Fading factors, on the other hand, provide a way of gradually forgetting examples. This is usually done with a decay function that assigns a weight to each example in the entire stream or a large window. Older examples receive smaller weights and are gradually treated as less important by the learner. Popular fading factors include linear, exponential, polynomial, and chordal functions.

Drift Detectors

Apart from sliding windows, another group of techniques that allow to construct a stream classifier are *drift detectors*. Their task is to detect concept drift and alarm a base learner that its classifier should be rebuilt or updated. For example, when a detector signals a sudden change, an existing classifier can be discarded and replaced by a new one trained only on the most recent data.

Drift detectors are usually implemented using statistical tests based on *sequential analysis*, *process control charts*, or *monitoring differences between two distributions*. Detectors based on sequential analysis check whether the classification error calculated on the most recent instances is significantly different from its value calculated for range of older instances. Examples of sequential tests include CUSUM and the Page-Hinkley test (Gama 2010). Drift detectors based on control charts take inspiration from statistical techniques used in quality control during product manufacturing. In these approaches, each prediction a classifier makes is treated as a Bernoulli trail. Then, the number of classification errors can be modeled with a Binomial distribution, which in turn can be tested for significantly improbable changes. Examples from this group include algorithms such as DDM, EDDM, and EWMA (Gama et al. 2014). Finally, several detection methods use two subsets of the stream: a reference window and a sliding window of the most recent examples. If the distributions over these two windows are significantly different, a change is signaled, suggesting that only examples from the sliding window should be used to create a new model.

Single Classifiers

First proposals of stream classifiers concentrated on processing massive stationary data sets in constant time per example. Decision trees were one of the first algorithms to be adapted to meet these requirements using the Hoeffding bound. This bound states that with probability $1 - \delta$, the true mean of a random variable of range R will not differ from the estimated mean after n independent observations by more than:

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}}. \tag{1}$$

Using the Hoeffding bound, Domingos and Hulten (2000) proposed a classifier called very fast decision tree (VFDT). This algorithm incrementally induces a tree from a massive data stream, without the need for storing examples after they have been used to update the tree. Its key idea is the selection of the split attribute, which is realized differently than in static trees (e.g., C4.5). Instead of selecting the best attribute (in terms of a split evaluation function) after viewing all the examples, VFDT uses the Hoeffding bound to calculate the number of examples necessary to select the right split node with probability $1 - \delta$. From the theoretical point of view, recent studies have shown that other bounds, as the ▸ McDiarmid inequality, are more suitable depending on the assumptions made about the distribution of values of the split evaluation function.

Many enhancements to the basic VFDT algorithm, often called the Hoeffding tree, have been proposed. They include methods of limiting memory usage, the use of alternative bounds which requires less examples for each split node, approaches to dealing with numerical attributes, pruning mechanisms, and the use of sliding windows or drift detectors to adapt the algorithm to non-stationary settings (Gama 2010). Nevertheless, the VFDT algorithm paved the way for many other learning algorithms that use the Hoeffding bound to incrementally process massive datasets (Ditzler et al. 2015).

Several traditional incremental classifiers were also adapted to computational and concept drift requirements. An illustrative example could be learning neural networks. By abandoning the epoch protocol and presenting examples in a single pass, neural networks can be adapted to changing data streams. Bayesian methods can also learn incrementally and require constant memory. To add a forgetting mechanism to this group of algorithms, sliding windows are usually employed to “unlearn” the oldest examples. Similarly, nearest neighbor classifiers are naturally transformed to incremental versions with different techniques for selecting the limited subset of the most “useful” examples for accurate predictions. Rule-based algorithms were also adjusted to data stream environments, in fact, FLORA algorithms developed by Kubat and Widmer were one of the first classifiers to cope with concept drift (Deckert 2013). Other algorithms use a structure similar to a decision tree to create rules and rule-specific drift detectors to react to changes (Kosina and Gama 2015).

Ensembles

▶ Ensembles are easily adapted to non-stationary data streams. Due to their modular construction, they are capable of incorporating new data elements by introducing a new component into the ensemble, updating existing component classifiers, or changing weights in the aggregation phase. Ensembles are usually categorized into block-based and online approaches.

Most block-based ensembles periodically evaluate component classifiers with the newest data block and substitute the worst ensemble member with a new (candidate) classifier. Additionally, practically all proposed approaches work with fixed-sized blocks. A generic block-based ensemble scheme is presented in Algorithm 1.

For each block B_i, the weights of current component classifiers $C_j \in \mathcal{E}$ are calculated by a quality measure $Q()$, which depends on the particular algorithm. For instance, in Accuracy Weighted Ensemble (AWE), $Q()$ is realized as a version of the mean square error of the component classifier C_j calculated on the recent block B_i, which is compared to the error of a random classifier on the same block (Wang et al. 2003). In addition to component re-weighting, a candidate classifier C_c is built from the recent block B_i and added to the ensemble if the ensemble's size is not exceeded. If the ensemble is full, the candidate classifier C_c substitutes the weakest ensemble member. It is worth noting that some algorithms, e.g., Learn++.NSE (Ditzler et al. 2015), do not limit the number of component classifiers in order to react to recurring concepts. The label prediction for new examples is usually based on a weighted majority vote of component classifiers. Most block-based ensembles take advantage of batch learning algorithms as component classifiers. This is not the case for hybrid algorithms, like the Accuracy Updated Ensemble (Brzezinski and Stefanowski 2014), which updates classifiers after processing each block.

Algorithm 1 Generic block-based ensemble

Input: $\mathcal{S}$, data stream of examples partitioned into blocks of size d; k, number of ensemble members; $Q()$, classifier quality measure;
Output: $\mathcal{E}$, ensemble of k weighted classifiers

```
for all blocks $B_i \in \mathcal{S}$ do
    build and weight candidate classifier $C_c$ using $B_i$ and $Q()$;
    weight all classifiers $C_j$ in ensemble $\mathcal{E}$ using $B_i$ and $Q()$;
    if $|\mathcal{E}| < k$ then
        $\mathcal{E} \leftarrow \mathcal{E} \cup \{C_c\}$;
    else if $\exists j : Q(C_c) > Q(C_j)$ then
        replace weakest ensemble member with $C_c$;
    end if
end for
```

The origins of online stationary ensembles come from research on the Winnow algorithm and the Weighted Majority Algorithm (Littlestone and Warmuth 1994), which combine the predictions of several experts (classifiers) by majority voting. When the ensemble misclassifies an instance, the weights of the wrong experts are decreased by a user-specified coefficient. The Dynamic Weighted Majority (DWM) is an extension of this idea for drifting data streams (Kolter and Maloof 2007). It uses a set of incremental classifiers, which are generated by the same learning algorithm. When a new example is available, the final prediction is obtained as a weighted vote of all classifiers. The weights of all classifiers that misclassify the example are decreased in the same way as in the Weighted Majority Algorithm. However, DWM dynamically creates and deletes component classifiers in response to changes in classification performance. If the ensemble's overall prediction is incorrect, a new classifier is added to the ensemble.

Another group of online ensembles includes generalizations of static ensembles. The most well known are online versions of ▶ bagging and ▶ boosting (Oza and Russell 2001). In case of online bagging, the key idea is to adapt the ▶ bootstrap sampling step to a streaming setting. This is done by using single examples multiple times according to the Poisson distribution.

This proposal of randomly updating training sets was an inspiration to develop several other approaches, e.g., leveraging bagging, online boosting, or the DDD ensemble (Ditzler et al. 2015).

Comprehensive reviews of various ensembles can be found in Ditzler et al. (2015), Gama (2010), and Kuncheva (2004).

Other Approaches

Although developing classifiers for concept-drifting streams is in itself a nontrivial task, some other characteristics of learning problems can make this task even more difficult. In most current algorithms, it is assumed that all information, in particular class labels of instances, are complete, immediately available, and received for free (Krempl et al. 2014). However, these assumptions may not hold true in some real-world problems, e.g., in fraud detection or patient health monitoring, where the labeling of examples is scarce or missing. In the case of static data, these problems are studied with ▸ semi-supervised learning. For adapting such techniques to streams, the availability of at least some labeled data from the most recent distribution is required. For instance, Masud et al. (2008) divide the stream into blocks containing partly labeled examples and then propose various approaches to combine learning ensemble classifiers with semi-supervised clustering. ▸ Active learning is also often related to semi-supervised frameworks. However, many sampling techniques developed for static data are not well suited for non-stationary streams (Spiliopoulou and Krempl 2013). A review of recent active learning strategies is presented in Žliobaitė et al. (2011).

A particularly challenging problem is learning classifiers from initially labeled non-stationary streams, where completely labeled examples are available for the first period only, followed by unlabeled data which may be drawn from a different distribution. Research on this topic is still at an early stage. Yet another problem is dealing with delayed information. In the case of *verification latency*, the class labels of preceding examples are not available before the subsequent instance has to be predicted. Therefore, feedback from correct predictions cannot be instantly used to improve the classifier. For a review of approaches that try to deal with this problem, see Ditzler et al. (2015).

Dealing with the ▸ class imbalance problem in non-stationary streams also introduces additional difficulties. Recent proposals to this problem pay attention to drifts of the minority class and specialized evaluation methods (Wang et al. 2015). The problem of class imbalance is also related to an increasing interest in studying other types of changes (Gama et al. 2014). Finally, other research concerns more complex representations of instances in streams, as graphs, semi-structured documents, or text messages, as well as complex target outputs, like multi-labeled or ordinal classification. Other open issues are discussed in Ditzler et al. (2015) and Krempl et al. (2014).

Applications

Applications of stream classification can be organized into three groups: monitoring and control, information management, and analytics and diagnostics (Zliobaite et al. 2015).

Monitoring and control mostly relates to the detection of abnormal events. Domains from this group include sensor networks, telecommunications, traffic control, and fraud detection. Information management encompasses applications such as product recommendation, crime prediction, personalized search, and customer profiling. Analytics and diagnostics address domains like evaluation of creditworthiness, budget planning, or drug resistance prediction.

Each of the aforementioned groups differs also in the way stream classification is modeled. Monitoring and control usually involves sequential data where the task is to detect sudden changes. Information management is mostly based on relational data and gradual rather than abrupt changes are to be expected. Finally, diagnostic applications often involve recurring concepts. For an in-depth analysis of different application settings, see Zliobaite et al. (2015).

Cross-References

- ▶ Classification
- ▶ Concept Drift
- ▶ Incremental Learning
- ▶ Online Learning

Recommended Reading

Aggarwal CC (ed) (2007) Data streams – models and algorithms. Volume 31 of Advances in database systems. Springer, New York

Bifet A, Gavaldà R (2007) Learning from time-changing data with adaptive windowing. In: Proceedings of the 7th SIAM international conference on data mining, Minneapolis, pp 443–448

Bifet A, Holmes G, Kirkby R, Pfahringer B (2010) MOA: massive online analysis. J Mach Learn Res 11:1601–1604

Brzezinski D, Stefanowski J (2014) Reacting to different types of concept drift: the accuracy updated ensemble algorithm. IEEE Trans Neural Netw Learn Syst 25:81–94

Deckert M (2013) Incremental rule-based learners for handling concept drift: an overview. Found Comput Decis Sci 38(1):35–65

Ditzler G, Roveri M, Alippi C, Polikar R (2015) Learning in nonstationary environments: a survey. IEEE Comput Intell Mag 10(4):12–25

Domingos P, Hulten G (2000) Mining high-speed data streams. In: Proceedings of the 6th ACM SIGKDD international conference on knowledge discovery and data mining, Boston, pp 71–80

Gama J (2010) Knowledge discovery from data streams. Chapman and Hall/CRC, Boca Raton

Gama J, Žliobaitė I, Bifet A, Pechenizkiy M, Bouchachia A (2014) A survey on concept drift adaptation. ACM Comput Surv 46(4):44:1–44:37

Gomes JB, Gaber MM, Sousa PAC, Ruiz EM (2014) Mining recurring concepts in a dynamic feature space. IEEE Trans Neural Netw Learn Syst 25(1):95–110

Kolter JZ, Maloof MA (2007) Dynamic weighted majority: an ensemble method for drifting concepts. J Mach Learn Res 8:2755–2790

Kosina P, Gama J (2015) Very fast decision rules for classification in data streams. Data Min Knowl Discov 29(1):168–202

Krempl G, Žliobaitė I, Brzezinski D, Hüllermeier E, Last M, Lemaire V, Noack T, Shaker A, Sievi S, Spiliopoulou M, Stefanowski J (2014) Open challenges for data stream mining research. SIGKDD Explor 16(1):1–10

Kuncheva LI (2004) Classifier ensembles for changing environments. In: Proceedings of 5th international workshop on multiple classifier systems, MCS 04, Cagliari. Volume 3077 of Springer LNCS, pp 1–15

Littlestone N, Warmuth MK (1994) The weighted majority algorithm. Inf Comput 108(2):212–261

Masud M, Gao J, Khan L, Thuraisingham B (2008) A practical approach to classify evolving data streams: training with limited amount of labeled data. In: Proceedings of the 8th IEEE international conference on data mining, Pisa, pp 929–934

Oza NC, Russell SJ (2001) Experimental comparisons of online and batch versions of bagging and boosting. In: Proceedings of the 7th ACM SIGKDD international conference on knowledge discovery and data mining, San Francisco, pp 359–364

Spiliopoulou M, Krempl G (2013) Tutorial mining multiple threads of streaming data. In: Proceedings of the Pacific-Asia conference on knowledge discovery and data mining, PAKDD 2013, Gold Coast

Wang H, Fan W, Yu PS, Han J (2003) Mining concept-drifting data streams using ensemble classifiers. In: Proceedings of the 9th ACM SIGKDD international conference on knowledge discovery and data mining, Washington, DC, pp 226–235

Wang S, Minku L, Yao X (2015) Resampling-based ensemble methods for online class imbalance learning. IEEE Trans Knowl Data Eng 27(5):1356–1368

Žliobaitė I, Bifet A, Pfahringer B, Holmes G (2011) Active learning with evolving streaming data. In: Proceedings of the 2011 European conference on machine learning and knowledge discovery in databases, Athens. Volume 6913 of Springer LNCS. pp 597–612

Zliobaite I, Pechenizkiy M, Gama J (2015) An overview of concept drift applications. In: Japkowicz N, Stefanowski J (eds) Big data analysis: new algorithms for a new society. Springer, Cham, pp 91–114

Stream Mining

A subfield of knowledge discovery called *stream mining* addresses the issue of rapidly changing data. The idea is to be able to deal with the stream of incoming data quickly enough to be able to simultaneously update the corresponding models (e.g., ontologies), as the amount of data is too large to be stored: new evidence from the incoming data is incorporated into the model without storing the data. For instance, modeling ontology changes and evolution over time using text mining methods (TextMining for Semantic Web). The underlying methods are based on the machine learning methods of ▶ Online Learning, where the model is built from the initially available data and updated regularly as more data become available.

Examples of data streams include computer network traffic, phone conversations, ATM transactions, web searches, and sensor data.

Cross-References

▶ Clustering from Data Streams
▶ Online Learning

String Kernel

A *string kernel* is a function from any of various families of kernel functions (see ▶ kernel methods) that operate over strings and sequences. The most typical example is as follows. Suppose that we are dealing with strings over a finite alphabet Σ. Given a string $a = a_1a_2\ldots a_n \in \Sigma^*$, we say that a substring $p = p_1p_2\ldots p_k$ *occurs* in a on positions $i_1i_2\ldots i_k$ iff $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ and $a_{i_j} = p_j$ for all $j = 1,\ldots,k$. We define the *weight* of this occurrence as $\lambda^{i_k - i_i - k + 1}$, where $\lambda \in [0,\ 1]$ is a constant chosen in advance; in other words, an occurrence weighs less if characters of p are separated by other characters. Let $\phi_p(a)$ be the sum of the weights of all occurrences of p in a, and let $\phi(a) = (\phi_p(a))_{p\in\sum^*}$ be an infinite-dimensional feature vector consisting of $\phi_p(a)$ for all possible substrings $p \in \Sigma^*$. It turns out that the dot product of two such infinite-length vectors, $K(a,a') = \phi(a)^T\phi(a')$, can be computed in time polynomial in the length of a and a', e.g., using dynamic programming. The function K defined in this way can be used as a kernel with various kernel methods. See also ▶ feature construction in text mining.

String Matching Algorithm

A string matching algorithm returns parts of text matching a given pattern, such as a *regular expression*. Such algorithms have countless applications, from file editing to bioinformatics. Many algorithms compute deterministic finite automata, which can be expensive to build, but are usually efficient to use; they include the *Knuth–Morris–Pratt* algorithm and the *Boyer–Moore* algorithm, that build the automaton in time $O(m)$ and $O(m + s)$, respectively, where m is the length of the pattern and s the size of the alphabet, and match a text of length n in time $O(n)$ in the worst case.

Structural Credit Assignment

▶ Credit Assignment

Structural Risk Minimization

Xinhua Zhang
NICTA, Australian National University,
Canberra, ACT, Australia
School of Computer Science, Australian
National University, Canberra, ACT, Australia
NICTA London Circuit, Canberra, ACT,
Australia

Abstract

Structural risk minimization is an inductive principle used to combat overfitting. It seeks a tradeoff between model complexity and fitness of the model on the training data.

Definition

The goal of learning is usually to find a model which delivers good generalization performance over an underlying distribution of the data. Consider an input space $\mathcal{X}$ and output space $\mathcal{Y}$. Assume the pairs $(X \times Y) \in \mathcal{X} \times \mathcal{Y}$ are random variables whose (unknown) joint distribution is

P_{XY}. It is our goal to find a predictor $f : \mathcal{X} \mapsto \mathcal{Y}$ which minimizes the expected risk:

$$P(f(X) \neq Y) = \mathbb{E}_{(X,Y)\sim P_{XY}}[\delta(f(X) \neq Y)],$$

where $\delta(z) = 1$ if z is true, and 0 otherwise.

In practice we only have n pairs of training examples (X_i, Y_i) drawn identically and independently from P_{XY}. Based on these samples, the ▶ Empirical Risk can be defined as

$$\frac{1}{n}\sum_{i=1}^{n}\delta(f(X_i) \neq Y_i).$$

Choosing a function f by minimizing the empirical risk often leads to ▶ Overfitting. To alleviate this problem, the idea of structural risk minimization (SRM) is to employ an infinite sequence of models $\mathcal{F}_1, \mathcal{F}_2, \ldots$ with increasing capacity. Here each $\mathcal{F}_i$ is a set of functions, e.g., polynomials with degree 3. We minimize the empirical risk in each model with a penalty for the capacity of the model:

$$f_n := \text{argmin}_{f\in\mathcal{F}_i, i\in\mathbb{N}} \frac{1}{n}\sum_{i=1}^{n}\delta(f(X_i) \neq Y_i)$$
$$+ \text{capacity}(\mathcal{F}_i, n),$$

where capacity$(\mathcal{F}_i, n)$ quantifies the complexity of model $\mathcal{F}_i$ in the context of the given training set. For example, it equals 2 when $\mathcal{F}_i$ is the set of polynomials with degree 2. In other words, when trying to reduce the risk on the training set, we prefer a predictor from a simple model.

Note the penalty is measured on the model $\mathcal{F}_i$, *not* the predictor f. This is different from the regularization framework, e.g., support vector machines, which penalizes the complexity of the *classifier*.

More details about SRM can be found in Vapnik (1998).

Recommended Reading

Vapnik V (1998) Statistical learning theory. John Wiley, New York

Structure

▶ Topology of a Neural Network

Structured Data Clustering

▶ Graph Clustering

Structured Induction

Michael Bain
University of New South Wales, Sydney, NSW, Australia

Definition

Structured induction is a method of applying machine learning in which a model for a task is learned using a representation where some of the components are themselves the outputs of learned models for specified sub-tasks. The idea was inspired by structured programming (Dahl et al. 1972), in which a complex task is solved by repeated decomposition into simpler sub-tasks that can be easily analyzed and implemented. The approach was first developed by Alen Shapiro (1987) in the context of constructing expert systems by ▶ decision tree learning, but in principle it could be applied using other learning methods.

Motivation and Background

Structured induction is designed to solve complex learning tasks for which it is difficult a priori to obtain a set of attributes or features in which it is possible to represent an accurate approximation of the target hypothesis reasonably concisely. In Shapiro's approach, a hierarchy of ▶ decision trees is learned, where in each tree of the hierarchy the attributes can have values that are outputs computed by a lower-level ▶ decision tree. Shapiro showed in computer chess applications that structured induction could learn accurate models, while significantly reducing their complexity. Structured induction was first commercialized in the 1980s by a number of companies providing expert systems solutions and has since seen many applications (Razzak et al. 1984).

A key assumption is that human expertise is available to define the task structure. Several approaches have been proposed to address the problem of learning this structure (under headings such as ▸ constructive induction, representation change, ▸ feature construction, and ▸ predicate invention) although to date, none have received wide acceptance.

The identification of knowledge acquisition as the "bottleneck" in knowledge engineering by Feigenbaum (1977) sparked considerable interest in symbolic machine-learning methods as a potential solution. Early work on ▸ decision tree induction around this time was often driven by problems from computer chess, a challenging domain by the standards of the time due to relatively large data sets and the complexity of the target hypotheses. In a landmark paper on his ID3 ▸ decision tree learning algorithm, Quinlan (1983) reported experiments on learning to classify positions in a four-piece chess endgame as winnable (or not) within a certain number of moves ("lost N-ply"). A set of attributes was defined as *inadequate* for a classification task if two objects belonging to different classes had identical values for each attribute. He concluded that "almost all the effort (for a non chess-player, at least) must be devoted to finding attributes that are adequate for the classification problem being tackled".

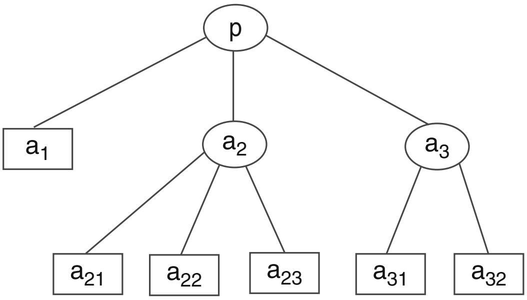
The problem is that the effort of developing the set of attributes becomes disproportionate to the time taken to do the induction. Quinlan (1983) reported durations of three weeks and two man-months, respectively, to define an adequate set of attributes for the "lost 2-ply" and "lost 3-ply" experiments. In contrast, the implementation of ID3 used in that work induced the ▸ decision trees in 3 s and 34 s, respectively. It is worth noting that the more complex problem of "lost 4-ply" was abandoned due to the difficulty of developing an adequate set of attributes.

Although Quinlan's experiments with ID3 produced exact ▸ classifiers for his chess problems, the resulting ▸ decision trees were too large to be comprehensible to domain experts. This is a serious drawback when machine learning is used with the goal of installing learned rules in an expert system, since the system cannot provide understandable explanations. Shapiro and Niblett (1982) proposed structured induction as a solution to this problem, and the method was developed in Shapiro's PhD thesis (Shapiro 1987) motivated by expert systems development.

Structure of Learning System

Structured induction is essentially a two-stage process, comprising a top-down decomposition of the problem into a solution structure, followed by a bottom-up series of ▸ classifier learning steps, one for each of the subproblems. A knowledge engineer and a domain expert are required to collaborate at each stage, with the latter acting as a source of examples. The use of machine learning to avoid the knowledge acquisition bottleneck is based on the finding that although domain experts find it difficult to express general and accurate rules for a problem, they are usually able to generate tutorial examples in an attribute-value formalism from which rules can be generalized automatically. The key insight of structured induction is that the task of specifying an attribute and its value set can be treated as a subproblem of the learning task, and solved in the same way.

The approach can be illustrated by a simple example using the structure shown in Fig. 1. Suppose the task is to learn a model for some concept $\mathbf{p}$. Suppose further that the domain expert proposes three attributes $\mathbf{a_1}$, $\mathbf{a_2}$, and $\mathbf{a_3}$ as adequate for the classification of $\mathbf{p}$. Now the domain expert consults with the knowledge engineer and it is decided that while attribute $\mathbf{a_1}$ is directly implementable, the other two are not. An attribute that is directly implementable by a knowledge engineer is referred to as *primitive* for the domain. The other attributes become sub-concepts $\mathbf{a_2}$ and $\mathbf{a_3}$, and each in turn is addressed by the domain expert. In this case, three attributes are proposed



Structured Induction, Fig. 1 A schematic diagram of a model learned by structured induction (after Shapiro 1987). Concepts to be learned are shown in ovals, and primitive attributes in boxes. The top-level concept **p** is defined in terms of the primitive attribute $\mathbf{a_1}$ and two sub-concepts $\mathbf{a_2}$ and $\mathbf{a_3}$. Each of the two sub-concepts are further decomposed into sets of primitive attributes, $\mathbf{a_{21\ldots3}}$ and $\mathbf{a_{31\ldots2}}$

as most relevant to the solution of $\mathbf{a_2}$, and two for $\mathbf{a_3}$. Since all of these attributes are found to be primitive, the top-down problem decomposition stage is therefore complete.

The domain expert, having proposed a set of primitive attributes for a sub-concept, say $\mathbf{a_3}$, is now required to provide a set of classified examples defined in terms of values for attributes $\mathbf{a_{31}}$ and $\mathbf{a_{32}}$. Given these examples, the knowledge engineer will run a learning algorithm to induce a ▸ classifier such as a ▸ decision tree. The domain expert will then inspect the ▸ classifier and can optionally refine it by supplying further examples, until they are satisfied that it completely and correctly defines the sub-concept $\mathbf{a_3}$. This process is repeated in a bottom-up fashion for all sub-concepts. At every level of the hierarchy, once all sub-concepts have been defined, they are now directly executable ▸ classifiers and can be treated in the same way as primitive attributes and used for learning. The structured induction solution is complete once an acceptable ▸ classifier has been learned for the top-level concept, **p** in this example.

Structured Versus Unstructured Induction

On two chess end-game domains, Shapiro (1987) showed that structured induction could generate more compact trees from fewer examples compared with an unstructured approach. To quantify this improvement, Shapiro made an analysis based on Michie's finite message information theory (Michie 1982). This showed that on one of the domains, the information gain contributed by the structured induction approach over learning unstructured trees from the same set of examples was 84 %. Essentially, this is because the structure devised by the domain expert in collaboration with the knowledge engineer provides a context for each of the induction tasks required. Since within this context only a subset of the complete attribute set is used to specify a sub-concept, it suffices to obtain only sufficient examples to learn a model for that sub-concept. However, without the benefit of such contextual restrictions the task of learning a complete solution can require considerably more examples. Shapiro's analysis is an attempt to quantify the relative increase in information per example in structured versus unstructured induction.

Related Work

While induction can bypass the knowledge acquisition bottleneck, in structured induction the process of acquiring the structure in collaboration with a domain expert can become a new bottleneck. In an attempt to avoid this, a number of researchers have attempted to develop methods whereby the structure, as well as the sub-concept models can be learned automatically.

Muggleton (1987) introduced ▶ inverse resolution as an approach to learning structured ▶ rule sets in a system called Duce. As part of this process, a domain expert is required to provide names for new sub-concepts or *predicates* that are proposed by the learning algorithm. A domain expert is also required to confirm learned rules. Both these roles are similar to those required of the expert in constructive induction, but the key difference is that the learning algorithm is now the source of both the structure and the rules. Duce was applied to one of the chess end-game domains used in Shapiro's study (Shapiro 1987) and found a solution that was less compact, but still accepted as comprehensible by a chess expert.

The Duce system searches for commonly occurring subsets of attribute-value pairs within rules, and uses these to construct new sub-concept definitions. Many approaches have been developed using related methods to learn new sub-concepts in the context of ▶ decision tree or ▶ rule learning; some examples include Pagallo and Haussler (1990), Zheng (1995), and Zhang and Honavar (2003). Gaines (1996) proposed EDAGs (exception directed acyclic graphs) as a generalization of both ▶ decision trees and rules with exceptions, and reported EDAG representations of chess end-game ▶ classifiers that were more comprehensible than either rules or ▶ decision trees. Zupan et al. (1999) developed a system named HINT designed to learn a model represented as a concept hierarchy based on methods of function decomposition. Inverse resolution as used in Duce has been generalized to first-order logic representations in the field of inductive logic programming. In this framework, the construction of new intermediate concepts is referred to as ▶ *predicate invention*, but to date this remains a largely open problem. More recently, much of the interest in representation change has focused on approaches like support vector machines, where the so-called kernel trick enables the use of *implicit* ▶ feature construction (Shawe-Taylor and Cristianini 2004).

Cross-References

▶ Classifier Systems
▶ Constructive Induction
▶ Decision Tree
▶ Feature Construction in Text Mining
▶ Predicate Invention
▶ Rule Learning

Recommended Reading

Dahl OJ, Dijkstra EW, Hoare CAR (eds) (1972) Structured programming. Academic Press, London

Feigenbaum EA (1977) The art of artificial intelligence: themes and case studies of knowledge engineering. In: Reddy R (ed) Proceedings of the fifth international conference on artificial intelligence (IJCAI'77). William Kaufmann, Los Altos, pp 1014–1029

Gaines B (1996) Transforming rules and trees into comprehensible knowledge structures. In: Fayyad U, Piatetsky-Shapiro G, Smyth P, Uthurusamy R (eds) Advances in knowledge discovery and data mining. MIT Press, Cambridge, MA, pp 205–226

Michie D (1982) Measuring the knowledge-content of expert programs. Bull Inst Math Appl 18(11/12):216–220

Muggleton S (1987) Duce, an oracle-based approach to constructive induction. In: IJCAI'87. Morgan Kaufmann, Los Altos, pp 287–292

Pagallo G, Haussler D (1990) Boolean feature discovery in empirical learning. Mach Learn 5:71–99

Quinlan JR (1983) Learning efficient classification procedures and their application to chess end games. In: Michalski R, Carbonnel J, Mitchell T (eds) Machine learning: an artificial intelligence approach. Tioga, Palo Alto, pp 464–482

Razzak MA, Hassan T, Pettipher R (1984) Extran-7: a Fortran-based software package for building expert systems. In: Bramer MA (ed) Research and development in expert systems. Cambridge University Press, Cambridge, pp 23–30

Shapiro A, Niblett T (1982) Automatic induction of classification rules for a chess endgame. In: Clarke MRB (ed) Advances in computer chess, vol 3. Pergamon Press, Oxford, pp 73–91

Shapiro AD (1987) Structured induction in expert systems. Turing Institute Press with Addison Wesley, Wokingham

Shawe-Taylor J, Cristianini N (2004) Kernel methods for pattern analysis. Cambridge University Press, Cambridge

Zhang J, Honavar V (2003) Learning decision tree classifiers from attribute value taxonomies and partially specified data. In: ICML-2003: Proceedings of the twentieth international conference on machine learning. AAAI Press, Menlo Park

Zheng Z (1995) Constructing nominal X-of-N attributes. In: Proceedings of the fourteenth International joint conference on artificial intelligence (IJCAI'95). Morgan Kaufmann, Los Altos, pp 1064–1070

Zupan B, Bohanec M, Demsar J, Bratko I (1999) Learning by discovering concept hierarchies. Artif Intell 109:211–242

Subgroup Discovery

Definition

Subgroup discovery (Klösgen 1996; Lavrač et al. 2004) is an area of ► supervised descriptive rule induction. The subgroup discovery task is defined as given a population of individuals and a property of those individuals that we are interested in, find population subgroups that are statistically "most interesting," for example, are as large as possible and have the most unusual statistical (distributional) characteristics with respect to the property of interest.

Recommended Reading

Klösgen W (1996) Explora: a multipattern and multistrategy discovery assistant. In: Advances in knowledge discovery and data mining. MIT Press, Cambridge, pp 249–271

Lavrač N, Kavšek B, Flach PA, Todorovski L (2004) Subgroup discovery with CN2-SD. J Mach Learn Res 5:153–188

Sublinear Clustering

Artur Czumaj[1] and Christian Sohler[2]
[1]University of Warwick, Coventry, UK
[2]University of Paderborn, Paderborn, Germany

Definition

Sublinear clustering describes the process of clustering a given set of input objects using only a small subset of the input set, which is typically selected by a random process. A solution computed by a sublinear clustering algorithm is an implicit description of the clustering (rather than a partition of the input objects), for example in the form of cluster centers. Sublinear clustering is usually applied when the input set is too large to be processed with standard clustering algorithms.

Motivation and Background

► Clustering is the process of partitioning a set of objects into subsets of similar objects. In machine learning, it is, for example, used in unsupervised learning to fit input data to a density model. In many modern applications of clustering, the input sets consist of billions of objects to be clustered. Typical examples include web search, analysis of web traffic, and spam detection. Therefore, even though many relatively efficient clustering algorithms are known, they are often too slow to cluster such huge inputs.

Since in some applications it is even not possible to cluster the entire input set, a new approach is needed to cope with very large data sets. The approach used in many different areas of science and engineering in this context is to *sample* a subset of the data and to analyze this sample instead of the whole data set. This is also the underlying method used in sublinear clustering. The main challenge and innovation in this area lies in the qualitative analysis of random sampling (in the form of approximation guarantees) and the design of *non uniform sampling* strategies that approximate the input set provably better than *uniform random sampling*.

Structure of the Learning System

In sublinear clustering a large input set of objects is to be partitioned into subsets of similar objects. Usually, this input is a point set P either in the Euclidean space or in the metric space. The clustering problem is specified by an objective function that determines the quality or cost of every possible clustering. The goal is to find a clustering of minimum cost/maximum quality. For example, given a set P of points in the Euclidean space the objective of ▶ *k-means clustering* is to find a set C of k centers that minimizes $\sum_{p \in P}(d(p, C))^2$, where $d(p, C)$ denotes the distance of p to the nearest center from C. Since usually the clustering problems are computationally hard ($\mathcal{NP}$-hard), one typically considers *approximate* solutions: instead of finding a clustering that optimizes the cost of the solution, one aims at a solution whose cost is close to the optimal one.

In sublinear algorithms a solution is computed for a small representative subset of objects, for example a random sample. The solution is represented implicitly, for example, in the form of cluster centers and it can be easily extended to the whole input point set. The quality of the output is analyzed *with respect to the original point set*. The challenge is to *design and analyzefast (sublinear-time) algorithms* that select a subset of objects that very well represent the entire input, such that the solution computed for this subset will also be a good solution for the original point set. This can be achieved by uniform and non uniform *random sampling* and the computation of *core-sets*, i.e., small weighted subsets of the input that approximate the input with respect to a clustering objective function.

Theory/Solution

Clustering Problems

For any point p and any set Q in a metric space (X, d), let $d(p, Q) = \min_{q \in Q} d(p, q)$. A point set P is *weighted* if there is a function w assigning a positive weight to each point in P.

Radius k-Clustering: Given a weighted set P of points in a metric space (X, d), find a set $C \subseteq P$ of k centers minimizing $\max_{p \in P} d(p, C)$.

Diameter k-Clustering: Given a weighted set P of points in a metric space (X, d), find a partition of P into k subsets $P_1, \ldots, P_k$, such that $\max_{i=1}^{k} \max_{p,q \in P_i} d(p, q)$ is minimized.

k-Median: Given a weighted set P of points in a metric space (X, d), find a set $C \subseteq P$ of k centers that minimizes $\mathit{median}(P, C) = \sum_{p \in P} w(p) \cdot d(p, C)$.

k-Means: Given a weighted set of points P in a metric space (X, d), find a set $C \subseteq P$ of k centers that minimizes $\mathit{mean}(P, C) = \sum_{p \in P} w(p) \cdot (d(p, C))^2$.

Random Sampling and Sublinear-Time Algorithms

Random sampling is perhaps the most natural approach to design sublinear-time algorithms for clustering. For the input set P, random sampling algorithm follows the following scheme:

1. Pick a random sample S of points
2. Run an algorithm (possibly an approximation one) for (given kind of) clustering for S
3. Return the clustering induced by the solution for S

The running time and the quality of this algorithm depends on the size of the random sample S and of the running time and the quality of the algorithm for clustering of S. Because almost all interesting clustering problems are computationally intractable ($\mathcal{NP}$-hard), usually the second step of the sampling scheme uses an approximation algorithm. (An algorithm for a minimization problem is called a λ-approximation if it always returns a solution whose cost is at most λ times the optimum.)

While random sampling approach gives very simple algorithms, depending on the clustering problem at hand, it often finds a clustering of poor quality and it is usually difficult to analyze. Indeed, it is easy to see that random sampling has some serious limitations to obtain clustering of good quality. Even the standard assumption that the input is in metric space is not sufficient to obtain good quality of clustering because of the small clusters which are "hidden" for random sampling approach. (If there is a cluster of size $o(|P|/|S|)$ then with high probability the random sample set S will contain no point from that cluster.) Therefore, another important parameters used in the analysis is the *diameter* of the metric space Δ, which is $\Delta = \max_{p,q \in P} d(p, q)$.

Quality of Uniformly Random Sampling: The quality of random sampling for three basic clustering problems (k-means, k-median, and min-sum k-clustering) have been analyzed in Ben-David (2004), Czumaj and Sohler (2007), and Mishra et al. (2001). In these papers, generic methods of analyzing uniform random sampling have been obtained. These results assume that the input point sets are in a metric space and are unweighted (i.e., when the weight function w is always 1).

Theorem 1 *Let* $\epsilon > 0$ *be an approximation parameter. Suppose that an* α*-approximation algorithm for the k-median problem in a metric space is used in step* (2) *of the uniform sampling, where* $\alpha \geq 1$ (Ben-David 2004; Czumaj and Sohler 2007; Mishra et al. 2001). *If we choose S to be of size at least*

$$c\alpha(K + \frac{\Delta}{\epsilon}(\alpha + k\ln(k\Delta\alpha/\epsilon)))$$

for an appropriate constant c, then the uniform sampling algorithm returns a clustering C^* *(of S) such that with probability at least 0.99 the normalized cost of clustering for S satisfies*

$$\frac{median(S, C^*)}{|S|} \leq \frac{2(\alpha + 0.01)OPT(P)}{|P|} + \epsilon,$$

where OPT(S) = min$_C$ *median(P,C) is the minimum cost of a solution for k-median for P.*

Similar results can be shown for the k-means problem, and also for min-sum k-clustering (cf. Czumaj and Sohler 2007). For example, for k-means, with a sample S of size at least $c\alpha(k + (\Delta^2/\epsilon)(\alpha + k\ln(k\Delta^2\alpha/\epsilon)))$, with probability at least 0. 99 the normalized cost of k-means clustering for S satisfy

$$\frac{mean(S, C^*)}{|S|^2} \leq \frac{4(\alpha + 0.01OPT(P))}{|P|^2} + \epsilon,$$

where $OPT(S) = \min_C\ mean(P, C)$ is the minimum cost of a solution for k-means for P.

Improvements of these bounds for the case when the input consists of points in Euclidean space are also discussed in Mishra et al. (2001) and Czumaj and Sohler (2007) discuss also. For example, for k-median, if one takes S of size at least $c\alpha(k + \Delta k\ln(\Delta/\epsilon)/\epsilon)$, then with probability at least 0.99 the normalized cost of k-median clustering for S satisfies

$$\frac{median(S, C^*)}{|S|} \leq \frac{(\alpha + 0.001)OPT(P)}{|P|} + \epsilon,$$

and hence the approximation ratio is better than that in Theorem 1 by factor of 2.

The results stated in Czumaj and Sohler (2007) allow to parameterize the constants 0.99 and 0.01 in the claims above.

Property Testing of the Quality of Clustering: Since most of the clustering problems are computationally quite expensive, in some situations it might be interesting not to find a clustering (or its succinct representation), but just to test if the input set has a good clustering.

Alon et al. (2003) introduced the notion of approximate testing of good clustering. A point set P is *c-clusterable* if it has a clustering of the cost at most c, that is, *OPT(P)* $\leq c$. To formalize the notion of having no good clustering, one says a point set is ε*-far from* $(1 + \beta)c$*-clusterable*, if more than an ε-fraction of the points in P must be removed (or moved) to make the input set $(1 + \beta)c$-clusterable. With these definitions, the goal is to design fast algorithms that accept the input point sets P, which are c-clusterable, and reject with probability at least 2/3 inputs are ε-far from $(1 + \beta)c$-clusterable. If neither holds, then the algorithms may either accept or reject. The bounds for the testing algorithms are phrased in terms of *sample complexity*, that is, the number of sampled input points which are considered by the algorithm (e.g., by using random sampling).

Alon et al. (2003) consider two classical clustering problems in this setting: radius and diameter k-clusterings. If the inputs are drawn from an arbitrary metric space, then they show that to distinguish between input points sets that are c-clusterable and are ε-far from $(1+\beta)c$-clusterable with $\beta < 1$, the sample complexity must be at least $\Omega(\sqrt{|P|/\epsilon})$. However, to distinguish between inputs that are c-clusterable and are ε-far from $2c$-clusterable, the sample complexity is only $O(\sqrt{k/\epsilon})$.

A more interesting situation is for the input points drawn from the Euclidean d-dimensional space. In that case, even a constant-time algorithms are possible.

Theorem 2 *For the radius k-clustering, one can distinguish between points sets in* R^d *that are c-clusterable from those ε-far from c-clusterable with the sample complexity* $\tilde{O}(dk/\epsilon)$ (Alon et al. 2003) *(The* $\tilde{O}$ *-notation ignores logarithms in the largest occurrence of a variable;* $\tilde{O}(f(n)) = O(f(n)\cdot(\log f(n))^{o(1)})$.*)*

Furthermore, for any $\beta > 0$*, one can distinguish between points sets in* R^d *that are c-clusterable from those ε-far from* $(1+\beta)c$*-clusterable with the sample complexity* $\tilde{O}(k^2/(\beta^2\epsilon))$.

Theorem 3 *For the diameter k-clustering, one can distinguish between points sets in* R^d *that are c-clusterable from those ε-far from* $(1+\beta)c$*-clusterable with the sample complexity* $\tilde{O}(k^2 d(2/\beta)^{2d}/\epsilon)$ (Alon et al. 2003).

Core-Sets: Sublinear Space Representations with Applications

A *core-set* is a small weighted set of points S that provably approximates another point set P with respect to a given clustering problem (Bădoiu et al. 2002). The precise definition of a core-set depends on the clustering objective function and the notion of approximation. For example, a coreset for the k-median problem can be defined as follows:

Definition 4 A weighted point set S is called ε-coreset for a point set P for the k-median problem, if for every set C of k centers, we have $(1-\epsilon)\cdot$ median $(P, C) \leq$ median $(S, C) \geq (1+\epsilon)\cdot$ median (P, C) (Har-Peled and Mazumdar 2004).

A core-set as defined above is also sometimes called a *strong* core-set, because the cost of the objective function is approximately preserved for any set of cluster centers. In some cases it can be helpful to only require a weaker notion of approximation. For example, for some applications it is sufficient that the cost is preserved for a certain discrete set of candidate solutions. Such a core-set is usually called a *weak* core-set. In high-dimensional applications it is sometimes sufficient that the solution is contained in the low-dimensional subspace spanned by the core-set points.

Constructing a Core-Set: There are deterministic and randomized constructions for core-sets of an unweighted set P of n points in the R^d. Deterministic core-set constructions are usually based on the *movement paradigm*. The input points are moved to a set of few locations such that the overall movement is at most ε times the cost of an optimal solution. Then the set of points at the same location are replaced by a single point whose weight equals the number of these points. Since for the k-median problem the cost of any solution changes by at most the overall movement, this immediately implies that the constructed weighted set is an ε-core-set. For other similar problems more involved arguments can be used to prove the core-set property. Based on the movement paradigm, for k-median a core-set of size $O(k \log n/\epsilon^d)$ can be constructed efficiently (Har-Peled and Mazumdar 2004).

Randomized core-set constructions are based on non uniform sampling. The challenge is to define a randomized process for which the resulting weighted point set is with high probability a core-set. Most randomized coreset constructions first compute a bi-criteria approximation C'. Then every point is sampled with probability proportional to its distance to the nearest center of C'. A point q is assigned a weight proportional to $1/p_q$, where p_q is the probability that p is sampled. For every fixed set C of k centers, the resulting sample is an unbiased estimator for *median*(P, C). If the sample set is large enough, it approximates the cost of every possible set of k centers within a factor of $(1 \pm \epsilon)$. The above approach can be used to obtain a weak core-set of size independent of the size of the input point set and the dimension of the input space (Feldman et al. 2007). A related construction has been previously used to obtain a strong core-set of size $\tilde{O}(k^2 \cdot d \cdot \log n/\epsilon^2)$. Both constructions have constant success probability that can be improved by increasing the size of the core-set.

Applications Core-sets have been used to obtain improved approximation algorithms for different variants of clustering problems. Since the core-sets are of sublinear size and they can be constructed in sublinear time, they can be used to obtain sublinear-time approximation algorithms for a number of clustering problems.

Another important application is clustering of data streams. A data stream is a long sequence of data that typically does not fit into main memory, for example, a sequence of packet headers in IP traffic monitoring. To analyze data streams we need algorithms that extract information from a stream without storing all of the observed data. Therefore, in the data streaming model algorithms are required to use $\log^{O(1)} n$ bits of memory. For core-sets, a simple but rather general technique is known, which turns a given construction of a strong core-set into a data streaming algorithm, i.e., an algorithm that processes the input points sequentially and uses only $\log^{O(1)}$ space (for constant k and ϵ) and computes a $(1 + \epsilon)$-approximation for the optimal set of centers of the k-median clustering (Har-Peled and Mazumdar 2004). Core-sets can also be used to improve the running time and stability of clustering algorithms like the k-means algorithm (Frahling and Sohler 2006).

Recommended Reading

Alon N, Dar S, Parnas M, Ron D (2003) Testing of clustering. SIAM J Discret Math 16(3):393–417

Bădoiu M, Har-Peled S, Indyk P (2002) Approximate clustering via core-sets. In: Proceedings of the 34th annual ACM symposium on theory of computing (STOC), Montreal, pp 250–257

Ben-David S (2004) A framework for statistical clustering with a constant time approximation algorithms for k-median clustering. In: Proceedings of the 17th annual conference on learning theory (COLT), Banff, pp 415–426

Chen K (2006) On k-median clustering in high dimensions. In: Proceedings of the 17th annual ACM-SIAM symposium on discrete algorithms (SODA), Miami, pp 1177–1185

Czumaj A, Sohler C (2007) Sublinear-time approximation for clustering via random sampling. Random Struct Algorithms 30(1–2):226–256

Feldman D, Monemizadeh M, Sohler C (2007) A PTAS for k-means clustering based on weak core-sets. In: Proceedings of the 23rd annual ACM symposium on computational geometry (SoCG), Gyeongju, pp 11–18

Frahling G, Sohler C (2006) A fast k-means implementation using coresets. In: Proceedings of the 22nd annual ACM symposium on computational geometry (SoCG), Sedona, pp 135–143

Har-Peled S, Kushal A (2005) Smaller coresets for k-median and k-means clustering. In: Proceedings of the 21st annual ACM symposium on computational geometry (SoCG), Pisa, pp 126–134

Har-Peled S, Mazumdar S (2004) On coresets for k-means and k-median clustering. In: Proceedings of the 36th annual ACM symposium on theory of computing (STOC), Chicago, pp 291–300

Meyerson A, O'Callaghan L, Plotkin S (2004) A k-median algorithm with running time independent of data size. Mach Learn 56(1–3):61–87

Mishra N, Oblinger D, Pitt L (2001) Sublinear time approximate clustering. In: Proceedings of the 12th annual ACM-SIAM symposium on discrete algorithms (SODA), Washington, DC, pp 439–447

Subspace Clustering

► Projective Clustering

Subsumption

Claude Sammut
The University of New South Wales, Sydney,
NSW, Australia

Subsumption provides a syntactic notion of generality. Generality can simply be defined in terms of the cover of a concept. That is, a concept, C, is more general than a concept, C', if C covers at least as many examples as C' (see ▶ Learning as Search). However, this does not tell us how to determine, from their syntax, if one sentence in a concept description language is more general than another. When we define a *subsumption* relation for a language, we provide a syntactic analogue of generality (Lavrač and Džeroski 1994). For example, θ*-subsumption* (Plotkin 1970) is the basis for constructing generalization lattices in ▶ inductive logic programming (Shapiro 1981). See ▶ Generality of Logic for a definition of θ*-subsumption*. An example of defining a subsumption relation for a domain specific language is in the LEX program (Mitchell et al. 1983), where an ordering on mathematical expressions is given.

Cross-References

▶ Generalization
▶ Induction
▶ Learning as Search
▶ Logic of Generality

Recommended Reading

Lavrač N, Džeroski S (1994) Inductive logic programming: techniques and applications. Ellis Horwood, Chichester

Mitchell TM, Utgoff PE, Banerji RB (1983) Learning by experimentation: acquiring and refining problem-solving heuristics. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning: an artificial intelligence approach. Tioga, Palo Alto

Plotkin GD (1970) A note on inductive generalization. In: Meltzer B, Michie D (eds) Machine intelligence, vol 5. Edinburgh University Press, Edinburgh, pp 153–163

Shapiro EY (1981) An algorithm that infers theories from facts. In: Proceedings of the seventh international joint conference on artificial intelligence, Vancouver. Morgan Kaufmann, Los Altos, pp 446–451

Supersmoothing

▶ Local Distance Metric Adaptation
▶ Locally Weighted Regression for Control

Supervised Descriptive Rule Induction

Petra Kralj Novak[1], Nada Lavrač[1,2], and Geoffrey I. Webb[3]
[1]Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia
[2]University of Nova Gorica, Nova Gorica, Slovenia
[3]Faculty of Information Technology, Monash University, Victoria, Australia

Synonyms

SDRI

Definition

Supervised descriptive rule induction (SDRI) is a machine learning task in which individual patterns in the form of rules (see ▶ classification rule) intended for interpretation are induced from data, labeled by a predefined property of interest. In contrast to standard ▶ supervised rule induction, which aims at learning a set of rules defining a classification/prediction model, the goal of SDRI is to induce individual descriptive patterns. In this respect, SDRI is similar to ▶ association rule discovery, but the consequents of the rules are restricted to a single variable – the property of interest – and, except for the discrete target attribute, the data is not necessarily assumed to be discrete.

Supervised descriptive rule induction assumes a set of training examples, described by attributes and their values and a selected attribute of interest (called the target attribute). Supervised descriptive rule induction induces rules that may each be interpreted independently of the others. Each rule is a local model, covering a subset of training examples, that captures a local relationship between the target attribute and the other attributes.

Induced descriptive rules are mainly aimed at human interpretation. More specifically, the purposes of supervised descriptive rule induction are to allow the user to gain insights into the data domain and to better understand the phenomena underlying the data.

Motivation and Background

Symbolic data analysis techniques aim at discovering comprehensible patterns or models in data. They can be divided into techniques for *predictive induction*, where models, typically induced from class-labeled data, are used to predict the class value of previously unseen examples, and *descriptive induction*, where the aim is to find comprehensible patterns, typically induced from unlabeled data. Until recently, these techniques have been investigated by two different research communities: predictive induction mainly by the machine learning community and descriptive induction mainly by the data mining community.

Data mining tasks where the goal is to find comprehensible patterns from labeled data have been addressed by both the machine learning and the data mining community independently. The data mining community, using the ▸ association rule learning perspective, adapted association rule learners like ▸ Apriori (Agrawal et al. 1996) to perform tasks on labeled data, like class association rule learning (Liu et al. 1998; Fürnkranz et al. 2012), as well as ▸ contrast set mining (Bay and Pazzani 2001) and ▸ emerging pattern mining (Dong and Li 1999). On the other hand, the machine learning community, which traditionally focused on the induction of ▸ rule sets from labeled data for the purposes of classification, turned to building individual rules for exploratory data analysis and interpretation. This is the goal of the task named ▸ subgroup discovery (Wrobel 1997). These are the main areas of supervised descriptive rule induction. All deal with finding comprehensible rules from class-labeled data. However, the methods used and the interpretation of the results differ slightly from approach to approach. Other related approaches include change mining, mining of closed sets for labeled data, exception rule mining, bump hunting, quantitative association rules, and impact rules. See Kralj Novak et al. (2009) for a more detailed survey of supervised descriptive rule induction.

Structure of the Learning System

Supervised descriptive rule induction assumes that there is data with the property of interest defined by the user. Let us illustrate supervised descriptive rule induction using data from Table 1, a very small artificial sample data set, adapted from Ross Quinlan (1986), which contains the results of a survey on 14 individuals, concerning the approval or disproval of an issue analyzed in the survey. Each individual is characterized by four attributes that encode rudimentary information about the sociodemographic background. The last column (Approved) is the designated property of interest, encoding whether the individual approved or disproved the issue. Unlike predictive induction, where the aim is to find a predictive model, the goal of supervised descriptive rule induction is to find local patterns in the form of individual rules describing individuals that are likely to approve or disprove the issue, based on the four demographic characteristics.

Figure 1 shows six descriptive rules, found for the sample data using the Magnum Opus (Webb 1995) rule learning software. These rules were found using the default settings except that the critical value for the statistical test was relaxed. This set of descriptive rules differs from a typical predictive rule set in several ways. The first rule

Supervised Descriptive Rule Induction, Table 1
A sample database

Education	Marital status	Sex	Has children	Approved
Primary	Single	Male	No	No
Primary	Single	Male	Yes	No
Primary	Married	Male	No	Yes
University	Divorced	Female	No	Yes
University	Married	Female	Yes	Yes
Secondary	Single	Male	No	No
University	Single	Female	No	Yes
Secondary	Divorced	Female	No	Yes
Secondary	Single	Female	Yes	Yes
Secondary	Married	Male	Yes	Yes
Primary	Married	Female	No	Yes
Secondary	Divorced	Male	Yes	No
University	Divorced	Female	Yes	No
Secondary	Divorced	Male	No	Yes

```
MaritalStatus=single AND Sex=male  →  Approved=no
Sex=male  →  Approved=no
Sex=female  →  Approved=yes
MaritalStatus=married  →  Approved=yes
MaritalStatus=divorced AND HasChildren=yes  →  Approved=no
MaritalStatus=single      Approved=no
```

Supervised Descriptive Rule Induction, Fig. 1 Selected descriptive rules, describing individual patterns in the data of Table 1

is redundant with respect to the second. The first rule is included as a strong pattern (all three single males do not approve), whereas the second is weaker but more general (four out of seven males do not approve, which is not highly predictive, but accounts for four out of all five respondents who do not approve). Most predictive systems would include only one of these rules, but either or both of them may be of interest to someone trying to understand the data, depending on the specific application. This particular approach to descriptive pattern discovery does not attempt to guess which of the more specific or more general patterns will be more useful to the end user. Another difference between predictive and descriptive rules is that the predictive approach often includes rules for the sake of completeness, while some descriptive approaches make no attempt at completeness, as they assess each pattern on its individual merits.

Exactly which rules will be induced by a supervised descriptive rule induction algorithm depends on the task definition, the selected algorithm, as well as the user-defined constraints concerning minimal rule support, precision, etc. Different learning approaches and heuristics have been proposed to induce supervised descriptive rules.

Applications

Applications of supervised descriptive rule induction are widely spread. See Kralj Novak et al. (2009) for a detailed survey.

Subgroup discovery has been used in numerous real-life applications Herrera et al. (2011). Medical applications include the analysis of coronary heart disease, brain ischemia data analysis, the analysis of cervical cancer, and psychiatric emergency, as well as profiling examiners for sonographic examinations. Spatial subgroup mining applications include mining of census data, mining of vegetation data and mining of demographic data. There are also applications in marketing, traffic accidents, production control, election analysis, and social data.

▶ Contrast set mining has been used with retail sales data and for designing customized insurance programs. It has also been used in medical applications to identify patterns in synchrotron x-ray data that distinguish tissue samples of different forms of cancerous tumor and for distinguishing between groups of brain ischemia patients.

▶ Emerging pattern mining has been mainly applied to the field of bioinformatics, more specifically to microarray data analysis. For example, an interpretable classifier based on simple rules that is competitive to the state of the art black-box classifiers on the acute lymphoblastic leukemia (ALL) microarray data set was built from emerging patterns. Another application was about finding groups of genes by emerging patterns in a ALL/AML data set and a colon tumor data set. Emerging patterns were also used together with the unexpected change approach and the added/perished rule to mine customer behavior.

Future Directions

A direction for further research is to decompose SDRI algorithms and preprocessing and evaluation methods into basic components and to reimplement them as connectable web services, which include the definition of interfaces between SDRI services. For instance, this can include the adaptation and implementation of subgroup discovery techniques to solving open problems in the area of contrast set mining and emerging patterns. This would allow for the improvement of algorithms due to the cross-fertilization of ideas from different SDRI subareas.

Another direction for further research concerns complex data types and the use of background knowledge. The SDRI attempts in this direction include relational subgroup discovery approaches like algorithms Midos (Wrobel 2001), RSD (relational subgroup discovery) (Železný and Lavrač 2006), and SubgroupMiner (Klösgen and May 2002), which is designed for spatial data mining in relational space databases. When ontologies are used as background knowledge to define the hypothesis search space and data are used to constrain and guide the hypothesis search and evolution, and this is called semantic subgroup discovery (Vavpetič and Lavrač 2013).

Cross-References

- ▶ Apriori Algorithm
- ▶ Association Rule
- ▶ Classification Rule
- ▶ Contrast Set Mining
- ▶ Emerging Patterns
- ▶ Rule Set
- ▶ Subgroup Discovery
- ▶ Supervised Learning

Recommended Reading

Agrawal R, Mannila H, Srikant R, Toivonen H, Verkamo AI (1996) Fast discovery of association rules. Advances in knowledge discovery and data mining. AAAI Press, Menlo Park, pp 307–328

Bay SD, Pazzani MJ (2001) Detecting group differences: mining contrast sets. Data Min Knowl Discov 5(3):213–246

Dong G, Li J (1999) Efficient mining of emerging patterns: discovering trends and differences. In: Proceedings of the 5th ACM SIGKDD international conference on knowledge discovery and data mining (KDD-99), pp 43–52

Fürnkranz J, Gamberger D, Lavrač N (2012) Foundations of rule learning. Springer, Heidelberg

Herrera F, Carmona C, Gonzlez P, del Jesus M (2011) An overview on subgroup discovery: foundations and applications. Knowl Info Syst 29(3):495–525

Klösgen W, May M (2002) Spatial subgroup mining integrated in an object-relational spatial database. In: Proceedings of the 6th European conference on principles and practice of knowledge discovery in databases (PKDD-02), pp 275–286

Kralj Novak P, Lavrač N, Webb GI (2009) Supervised descriptive rule discovery: a unifying survey of contrast set, emerging pattern and subgroup mining. J Mach Learn Res 10:377–403. http://www.jmlr.org/papers/volume10/kralj-novak09a/kralj-novak09a.pdf

Liu B, Hsu W, Ma Y (1998) Integrating classification and association rule mining. In: Proceedings of the 4th international conference on knowledge discovery and data mining (KDD-98), pp 80–86

Ross Quinlan J (1986) Induction of decision trees. Mach Learn 1(1):81–106

Vavpetič A, Lavrač N (2013) Semantic subgroup discovery systems and workflows in the SDM-toolkit. Comput J 56(3):304–320

Webb GI (1995) OPUS: an efficient admissible algorithm for unordered search. J Artif Intell Res 3:431–465

Wrobel S (1997) An algorithm for multi-relational discovery of subgroups. In: Proceedings of the 1st European conference on principles of data mining and knowledge discovery (PKDD-97), pp 78–87

Wrobel S (2001) Inductive logic programming for knowledge discovery in databases. In: Džeroski S, Lavrač N (eds) Relational data mining, chap 4. Springer, Berlin/New York, pp 74–101

Železný F, Lavrač N (2006) Propositionalization-based relational subgroup discovery with RSD. Mach Learn 62:33–63

Supervised Learning

Definition

Supervised learning refers to any machine learning process that learns a function from an input type to an output type using data comprising examples that have both input and output values. Two typical examples of supervised learning are ▶ classification learning and ▶ regression. In these cases, the output types are respectively categorical (the classes) and numeric. Supervised learning stands in contrast to ▶ unsupervised learning, which seeks to learn structure in data, and to ▶ reinforcement learning in which sequential decision-making policies are learned from reward with no examples of "correct" behavior.

Cross-References

▶ Reinforcement Learning
▶ Unsupervised Learning

Supervised Learning on Text Data

▶ Document Classification

Support Vector Machines

Xinhua Zhang
NICTA, Australian National University, Canberra, ACT, Australia
School of Computer Science, Australian National University, Canberra, ACT, Australia
NICTA London Circuit, Canberra, ACT, Australia

Abstract

Support vector machines (SVMs) are a class of linear algorithms which can be used for classification, regression, density estimation, novelty detection, etc. In the simplest case of two-class classification, SVMs find a hyperplane that separates the two classes of data with as wide a margin as possible. This leads to good generalization accuracy on unseen data and supports specialized optimization methods that allow SVM to learn from a large amount of data.

Motivation and Background

Over the past decade, maximum margin models especially SVMs have become popular in machine learning. This technique was developed in three major steps. First, assuming that the two classes of training examples can be separated by a hyperplane, Vapnik and Lerner proposed in 1963 that the optimal hyperplane is the one that separates the training examples with the widest margin. From the 1960s to 1990s, Vapnik and Chervonenkis developed the Vapnik-Chervonenkis theory, which justifies the maximum margin principle from a statistical point of view. Similar algorithms and optimization techniques were proposed by Mangasarian in (1965).

Second, Boser et al. (1992) incorporated kernel function into the maximum margin models, and their formulation is close to the currently popular form of SVMs. Before that, Wahba (1990) also discussed the use of kernels. Kernels allow SVM to implicitly construct the optimal hyperplane in the feature space, and the resulting nonlinear model is important for modeling real data.

Finally, in case the training examples are not linearly separable, Cortes and Vapnik (1995) showed that the soft margin can be applied, allowing some examples to violate the margin condition.

On the theoretical side, Shawe-Taylor et al. (1998) gave the first rigorous statistical bound on the generalization of hard-margin SVMs. Shawe-Taylor and Cristianini (2000) gave statistical bounds on the generalization of soft-margin algorithms and for the regression case.

In reality, SVMs became popular thanks to its significantly better empirical performance than the neural networks. By incorporating transform invariances, the SVMs developed at AT&T achieved the highest accuracy on the MNIST benchmark set (a handwritten digit recognition problem). Joachims (1998) also showed clear superiority of SVMs on text categorization. Afterwards, SVMs have been shown effective in many applications including computer vision, natural language, bioinformatics, finance, etc.

Theory

SVM has a stronger mathematical basis than some machine learning methods such as neural networks and is closely related to some well-established theories in statistics. As a linear model, it not only tries to correctly classify the training data but also maximizes the margin for better generalization performance. This formulation leads to a separating hyperplane that depends only on the (usually small fraction of) data points that lie on the margin, which are called support vectors. Hence the whole algorithm is called support vector machine. In addition, since real-world data analysis problems often involve nonlinear dependencies, SVMs can be easily extended to model such nonlinearity by means of positive semi-definite kernels. Moreover, SVMs can be trained via quadratic programming, which (a) makes theoretical analysis easier and (b) provides much convenience in designing efficient solvers that scale for large datasets. Finally, when applied to real-world data, SVMs often deliver state-of-the-art performance in accuracy, flexibility, robustness, and efficiency.

Optimal Hyperplane for Linearly Separable Examples

Consider the training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ where $\mathbf{x}_i \in \mathbb{R}^p$ is the input feature vector for the i-th example and $y_i \in \{1, -1\}$ is its corresponding label indicating whether the example is positive ($y_i = +1$) or negative ($y_i = -1$). To begin with, we assume that the set of positive and negative examples are linearly separable, i.e., there exists a function $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$ where $\mathbf{w} \in \mathbb{R}^p$ (called the weight vector) and $b \in \mathbb{R}$ (called bias) such that

$$\langle \mathbf{w}, \mathbf{x}_i \rangle + b > 0 \quad \text{for } y_i = +1$$

$$\langle \mathbf{w}, \mathbf{x}_i \rangle + b < 0 \quad \text{for } y_i = -1.$$

We call $\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$ the decision hyperplane, and in fact, there can exist multiple hyperplanes that separate the positive and negative examples; see Fig. 1. However, they are not created equal. Associated with each such hyperplane is a notion called *margin*, defined as the distance between the hyperplane and the closest example. SVM aims to find the particular hyperplane that maximizes the margin.

Mathematically, it is easy to check that the distance from a point $\mathbf{x}_i$ to a hyperplane $\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$ is $\|\mathbf{w}\|^{-1} |\langle \mathbf{w}, \mathbf{x}_i \rangle + b|$. Therefore, SVM seeks for the optimal $\mathbf{w}, b$ of the following optimization problem:

$$\underset{\mathbf{w}\in\mathbb{R}^p,\, b\in\mathbb{R}}{\text{maximize}} \ \min_{1\le i\le n} \frac{|\langle \mathbf{w}, \mathbf{x}_i \rangle + b|}{\|\mathbf{w}\|}, \quad s.t. \begin{cases} \langle \mathbf{w}, \mathbf{x}_i \rangle + b > 0 \text{ if } y_i = +1 \\ \langle \mathbf{w}, \mathbf{x}_i \rangle + b < 0 \text{ if } y_i = -1 \end{cases} \quad \forall i \ .$$

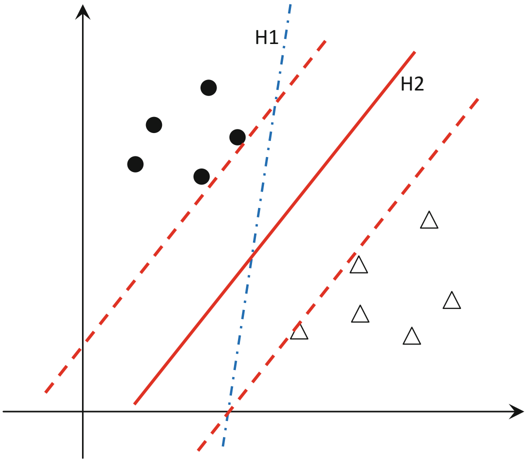
It is clear that if $(\mathbf{w}, b)$ is an optimal solution, then $(\alpha\mathbf{w}, \alpha b)$ is also an optimal solution for any $\alpha > 0$. Therefore, to fix the scale, we can equivalently set the numerator of the objective $\min_{1\le i\le n} |\langle \mathbf{w}, \mathbf{x}_i \rangle + b|$ to 1 and minimize the denominator $\|\mathbf{w}\|$:

$$\underset{\mathbf{w}\in\mathbb{R}^p,\, b\in\mathbb{R}}{\text{minimize}} \ \|\mathbf{w}\|^2, \quad s.t. \begin{cases} \langle \mathbf{w}, \mathbf{x}_i \rangle + b > 1 \quad \text{if } y_i = +1 \\ \langle \mathbf{w}, \mathbf{x}_i \rangle + b < -1 \text{ if } y_i = -1. \end{cases} \quad \forall i \ . \tag{1}$$

This is a linearly constrained quadratic program, which can be solved efficiently. Hence, it becomes the most commonly used (primal) form of SVM for the linearly separable case.

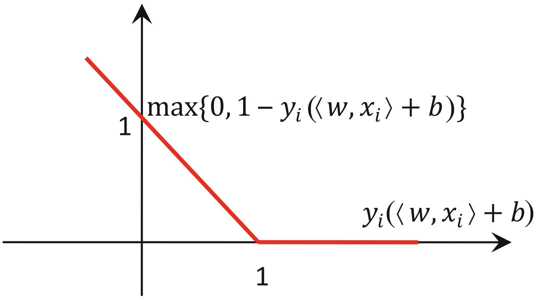
Soft Margins

In practice, most, if not all, datasets are not linearly separable, i.e., no $\mathbf{w}$ and b can satisfy the constraints of the optimization problem (1). In this case, we will allow some data points



Support Vector Machines, Fig. 1 Example of maximum margin separator. Both H1 and H2 correctly separate the examples from the two classes. But H2 has a wider margin than H1

to violate the margin condition and penalize it accordingly. Mathematically, notice that the constraints in (1) can be equivalently written as $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 1$. Now we introduce a new set of nonnegative slack variables ξ_i into the constraints:



Support Vector Machines, Fig. 2 Graph of hinge loss

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 1 - \xi_i \ ,$$

and incorporate a penalty into the original objective to derive the soft-margin SVM:

$$\underset{\mathbf{w}, b, \xi_i}{\text{minimize}} \ \lambda \|\mathbf{w}\|^2 + \frac{1}{n} \sum_{i=1}^{n} \xi_i \quad s.t. \ y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 1 - \xi_i, \text{and } \xi_i > 0 \ \ \forall i \ . \tag{2}$$

$\lambda > 0$ is a tradeoff factor. It is important to note that ξ_i can be written as $\xi_i = \max\{0, 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)\}$, which is called hinge loss and is depicted in Fig. 2. This way, the optimization problem can be reformulated into an unconstrained non-smooth problem:

$$\underset{\mathbf{w} \in \mathbb{R}^p, \, b \in \mathbb{R}}{\text{minimize}} \ \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)\} \ . \tag{3}$$

Notice that $\max\{0, 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)\}$ is also a convex upper bound of $\delta(y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 0)$, where $\delta(x) = 1$ if x is true and 0 otherwise. Therefore, the penalty we use is a convex upper bound of the average training error. When the training set is actually separable, the soft-margin problem (2) automatically recovers the hard-margin problem (1) when λ is sufficiently large.

Dual Forms and Kernelization

As the constraints in the primal form (2) are not convenient to handle, people have conventionally resorted to the dual problem of (2). Following the standard procedures, one can derive the Lagrangian dual

$$\frac{1}{2\lambda} \sum_{i,j} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \sum_i \alpha_i, \quad s.t. \ \alpha_i \in [0, n^{-1}], \ \text{and} \ \sum_i y_i \alpha_i = 0 \ . \tag{4}$$

which is again a quadratic program but with much simpler constraints: box constraints plus a single linear equality constraint. To recover the primal solution $\mathbf{w}^*$ from the dual solution α_i^*, we have

$$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \mathbf{x}_i \ ,$$

and the optimal bias b can be determined by using the duality conditions.

The training examples can be divided into three categories according to the value of α_i^*. If $\alpha_i^* = 0$, it means the corresponding training example does not affect the decision boundary, and in fact it lies beyond the margin, i.e., $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 1$. If $\alpha_i^* \in (0, n^{-1})$, then the training example lies on the margin, i.e., $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) = 1$. If $\alpha_i^* = n^{-1}$, it means the training example violates the margin, i.e., $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) < 1$. In the latter two cases where $\alpha_i^* > 0$, the i-th training example is called a support vector.

In many applications, most α_i^* in the optimal solution are 0, which gives a sparse solution. As the final classifier depends only on those support vectors, the whole algorithm is named support vector machines.

From the dual problem (4), a key observation can be drawn that the feature of the training examples $\{\mathbf{x}_i\}$ influences training only via the inner product $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$. Therefore, we can redefine the feature by mapping $\mathbf{x}_i$ to a richer feature space via $\phi(\mathbf{x}_i)$ and then compute the inner product there: $k(\mathbf{x}_i, \mathbf{x}_j) := \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$. Furthermore, one can even directly define k without explicitly specifying ϕ. This allows us to (a) implicitly use a rich feature space whose dimension can be infinitely high and (b) apply SVM to non-Euclidean spaces as long as a kernel $k(\mathbf{x}_i, \mathbf{x}_j)$ can be defined on it. Examples include strings and graphs (Haussler 1999), which have been widely applied in bioinformatics (Schölkopf et al. 2004). Mathematically, the objective (4) can be kernelized into

$$\frac{1}{2\lambda} \sum_{i,j} y_i y_j \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) - \sum_i \alpha_i, \quad s.t.\ \alpha_i \in [0, n^{-1}], \text{ and } \sum_i y_i \alpha_i = 0 \ . \tag{5}$$

However, now the $\mathbf{w}$ cannot be expressed just in terms of kernels because $\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \phi(\mathbf{x}_i)$. Fortunately, when predicting on a new example $\mathbf{x}$, we again only require the inner product and hence use kernel only:

$$\begin{aligned} \langle \mathbf{w}^*, \mathbf{x} \rangle &= \sum_{i=1}^{n} \alpha_i^* y_i \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle \\ &= \sum_{i=1}^{n} \alpha_i^* y_i k(\mathbf{x}_i, \mathbf{x}) \ . \end{aligned}$$

Commonly used kernels on $\mathbb{R}^n$ include polynomial kernels $(1 + \langle \mathbf{x}_i, \mathbf{x}_j \rangle)^d$, Gaussian RBF kernels $\exp(-\gamma \left\| \mathbf{x}_i - \mathbf{x}_j \right\|^2)$, Laplace RBF kernels $\exp(-\gamma \left\| \mathbf{x}_i - \mathbf{x}_j \right\|)$, etc. Kernels on strings and trees are usually based on convolution which requires involved algorithms for efficient evaluation (Haussler 1999; Borgwardt 2007). More details can be found in the kernel section.

Optimization Techniques and Toolkits

The main challenge of optimization lies in scaling for large datasets, i.e., n and p are large. Decomposition methods based on the dual problem is the first popularly used method for solving large-scale SVMs. For example, sequential minimal optimization (SMO) optimizes two dual variables α_i, α_j analytically in each iteration (Platt 1999a). An SMO-type implementation is available in the LibSVM package http://www.csie.ntu.edu.tw/~cjlin/libsvm. Another popular package using decomposition methods is the SVM-light, available at http://svmlight.joachims.org. Coordinate descent in the dual is also effective and converges at linear rate. An implementation can be downloaded from http://www.csie.ntu.edu.tw/~cjlin/liblinear.

Primal methods are also popular, most of which are based on formulating the objective as a non-smooth objective function like (3). An important type is the subgradient descent method, which is similar to gradient descent but uses a subgradient due to the non-smooth objective. When the dataset is large, one can further use a random subset of training examples to efficiently compute the (approximate) subgradient, and algorithms exist that guarantee the convergence in probability. This is called stochastic subgradient descent, and in practice, it can often quickly find a reasonably good solution. A package that implements this idea can be found at http://leon.bottou.org/projects/sgd.

Finally, cutting plane and bundle methods are also effective (Tsochantaridis et al. 2005; Smola et al. 2007), and they are especially useful for generalized SVMs with structured outputs. An implementation is the bundle method for risk minimization (BMRM), available for download at http://users.rsise.anu.edu.au/~chteo/BMRM.html.

Applications

The above description of SVM focused on binary class classification. In fact, SVMs, or the ideas of maximum margin and kernel, have been widely used in many other learning problems such as regression, ranking and ordinal regression, density estimation, novelty detection, quantile regression, etc. Even in classification, SVM has been extended to the case of multi-class, multi-label, and structured output (Tsochantaridis et al. 2005; Taskar 2004).

For multi-class classification and structured output classification where the possible label set $\mathcal{Y}$ can be large, maximum margin machines can be formulated by introducing a joint feature map ϕ on pairs of (x_i, y) $(y \in \mathcal{Y})$. Letting $\Delta(y_i, y)$ be the discrepancy between the true label y_i and the candidate label y, the primal form can be written as

$$\underset{w,\xi_i}{\text{minimize}} \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n}\xi_i, \quad s.t.\ \langle w, \phi(x_i, y_i) - \phi(x_i, y)\rangle \geq \Delta(y_i, y) - \xi_i,\ \forall\, i, y,$$

and the dual form is

$$\underset{\alpha_{i,y}}{\text{minimize}} \frac{1}{2\lambda} \sum_{(i,y),(i',y')} \alpha_{i,y}\alpha_{i',y'} \left\langle \phi(x_i, y_i) - \phi(x_i, y), \phi(x_{i'}, y_{i'}) - \phi(x_{i'}, y')\right\rangle - \sum_{i,y} \Delta(y_i, y)\alpha_{i,y}$$

$$s.t. \quad \alpha_{i,y} \geq 0,\ \forall\, i, y; \quad \sum_{y} \alpha_{i,y} = \frac{1}{n},\ \forall i.$$

Again kernelization is convenient, by simply replacing all the inner products $\langle \phi(x_i, y), \phi(x_{i'}, y')\rangle$ with a joint kernel $k((x_i, y), (x_{i'}, y'))$. Further factorization using graphical models is possible; see (Taskar 2004). Notice when $\mathcal{Y} = \{1, -1\}$, setting $\phi(x_i, y) = y\phi(x_i)$ recovers the binary SVM formulation. Effective methods to optimize the dual objective include SMO, exponentiated gradient descent, mirror descent, cutting plane, or bundle methods.

In general, SVMs are not trained to output the odds of class membership, although the posterior probability is desired to enable post-processing. Platt (1999b) proposed training an SVM and then training the parameters of an additional sigmoid function to map the SVM outputs into probabilities. A more principled approach is the relevance vector machine, which has an identical functional form to the SVMs and uses Bayesian inference to obtain sparse solutions for probabilistic classification.

As mentioned above, the hinge loss used in SVM is essentially a convex surrogate of the misclassification loss, i.e., 1 if the current weight **w** misclassifies the training example and 0 otherwise. Minimizing the misclassification loss is proved NP-hard, so for computational convenience, continuous convex surrogates are used, including hinge loss, exponential loss, and logistic loss. Their statistical properties are studied by Jordan et al. (2003). For hinge loss, it has the significant merit of sparsity in the dual, which leads to robustness and good generalization performance.

SVMs have been widely applied in real-world problems. In history, its first practical success was gained in handwritten digit recognition. By incorporating transform invariances, the SVMs developed at AT&T achieved the highest accuracy on the MNIST benchmark set. It has also been very effective in computer vision applications such as object recognition and detection. With the special advantage in handling high-dimensional data, SVMs have witnessed wide application in bioinformatics such as microarray processing (Schölkopf et al. 2004) and natural language processing like named entity recognition, part-of-speech tagging, parsing, and chunking (Taskar 2004; Joachims 1998).

Cross-References

▸ Kernel Methods
▸ Radial Basis Function Networks

Recommended Reading

A comprehensive treatment of SVMs can be found in Schölkopf and Smola (2002) and Shawe-Taylor and Cristianini (2004). Some important recent developments of SVMs for structured output are collected in Bakir et al. (2007). As far as applications are concerned, see Lampert (2009) for computer vision and Schölkopf et al. (2004) for bioinformatics. Finally, Vapnik (1998) provides the details on statistical learning theory.

Bakir G, Hofmann T, Schölkopf B, Smola A, Taskar B, Vishwanathan SVN (2007) Predicting structured data. MIT Press, Cambridge

Borgwardt KM (2007) Graph kernels. Ph.D. thesis, Ludwig-Maximilians-University, Munich

Boser B, Guyon I, Vapnik V (1992) A training algorithm for optimal margin classifiers. In: Haussler D (ed) Proceedings of annual conference on computational learning theory, Pittsburgh. ACM Press, pp 144–152

Cortes C, Vapnik V (1995) Support vector networks. Mach Learn 20(3):273–297

Haussler D (1999) Convolution kernels on discrete structures. Technical report UCS-CRL-99-10, UC Santa Cruz

Joachims T (1998) Text categorization with support vector machines: learning with many relevant features. In: Proceedings of the European conference on machine learning. Springer, Berlin, pp 137–142

Jordan MI, Bartlett PL, McAuliffe JD (2003) Convexity, classification, and risk bounds. Technical report 638, UC Berkeley

Lampert CH (2009) Kernel methods in computer vision. Found Trends Comput Graph Vis 4(3): 193–285

Mangasarian OL (1965) Linear and nonlinear separation of patterns by linear programming. Operations Research 13:444–452

Platt JC (1999a) Fast training of support vector machines using sequential minimal optimization. In: Advances in kernel methods—support vector learning. MIT Press, pp 185–208

Platt JC (1999b) Probabilities for sv machines. In: Smola AJ, Bartlett PL, Schölkopf B, Schuurmans D (eds) Advances in large margin classifiers. MIT Press, Cambridge, MA, pp 61–74

Schölkopf B, Smola A (2002) Learning with kernels. MIT Press, Cambridge, MA

Schölkopf B, Tsuda K, Vert J-P (2004) Kernel methods in computational biology. MIT Press, Cambridge, MA

Shawe-Taylor J, Bartlett PL, Williamson RC, Anthony M (1998) Structural risk minimization over data-dependent hierarchies. IEEE Trans Inf Theory 44(5):1926–1940

Shawe-Taylor J, Cristianini N (2000) Margin distribution and soft margin. In: Smola AJ, Bartlett PL, Schölkopf B, Schuurmans D (eds) Advances in large margin classifiers. MIT Press, Cambridge, MA, pp 349–358

Shawe-Taylor J, Cristianini N (2004) Kernel methods for pattern analysis. Cambridge University Press, Cambridge, UK

Smola A, Vishwanathan SVN, Le Q (2007) Bundle methods for machine learning. In: Koller D, Singer Y (eds) Advances in neural information processing systems, vol 20. MIT Press, Cambridge MA

Taskar B (2004) Learning structured prediction models: a large margin approach. Ph.D. thesis, Stanford University

Tsochantaridis I, Joachims T, Hofmann T, Altun Y (2005) Large margin methods for structured and interdependent output variables. J Mach Learn Res 6:1453–1484
Vapnik V (1998) Statistical learning theory. John Wiley, New York
Wahba G (1990) Spline models for observational data. CBMS-NSF regional conference series in applied mathematics, vol 59. SIAM, Philadelphia

Swarm Intelligence

Swarm intelligence is the discipline that studies the collective behavior of systems composed of many individuals that interact locally with each other and with their environment and that rely on forms of decentralized control and self-organization. Examples of such systems are colonies of ants and termites, schools of fish, flocks of birds, herds of land animals, and also some artifacts, including swarm robotic systems and some computer programs for tackling optimization problems such as ▶ ant colony optimization and ▶ particle swarm optimization.

Symbolic Dynamic Programming

Scott Sanner[1] and Kristian Kersting[2,3]
[1]Statistical Machine Learning Group, NICTA, Canberra, ACT, Australia
[2]Technische Universität Dortmund, Dortmund, Germany
[3]Knowledge Discovery, Fraunhofer IAIS, Sankt Augustin, Germany

Synonyms

Dynamic programming for relational domains; Relational dynamic programming; Relational value iteration; SDP

Definition

Symbolic dynamic programming (SDP) is a generalization of the ▶ dynamic programming technique for solving ▶ Markov decision processes (MDPs) that exploits the symbolic structure in the solution of relational and first-order logical MDPs through a lifted version of dynamic programming.

Motivation and Background

Decision-theoretic planning aims at constructing a policy for acting in an uncertain environment that maximizes an agent's expected utility along a sequence of steps. For this task, Markov decision processes (MDPs) have become the standard model. However, classical dynamic programming algorithms for solving MDPs require explicit state and action enumeration, which is often impractical: the number of states and actions grows very quickly with the number of domain objects and relations. In contrast, SDP algorithms seek to avoid explicit state and action enumeration through the symbolic representation of an MDP and a corresponding symbolic derivation of its solution, such as a value function. In essence, SDP algorithms exploit the symbolic structure of the MDP representation to construct a minimal logical partition of the state space required to make all necessary value distinctions.

Theory and Solution

Consider an agent acting in a simple variant of the BoxWorld problem. There are several cities such as *London*, *Paris* etc., trucks $truck_1$, $truck_2$ etc., and boxes box_1, box_2 etc. The agent can load a box onto a truck or unload it and can drive a truck from one city to another. Only when a particular box, say box box_1, is in a particular city, say *Paris*, the agent receives a positive reward. The agent's learning task is now to find a policy for action selection that maximizes its reward over the long term.

A great variety of techniques for solving such decision-theoretic planning tasks have been developed over the last decades. Most of them assume atomic representations, which essentially

- *Domain Object Types (i.e., sorts)*: $Box, Truck, City = \{paris, \ldots\}$
- *Relations (with parameter sorts)*:
  $BoxIn(Box, City), TruckIn(Truck, City), BoxOn(Box, Truck)$
- *Reward*: if $\exists b.BoxIn(b, paris)$ 10 else 0
- *Actions (with parameter sorts)*:
  - $load(Box : b, Truck : t, City : c)$:
    - Success Probability: if $(BoxIn(b, c) \wedge TruckIn(t, c))$ then .9 else 0
    - Add Effects on Success: $\{BoxOn(b, t)\}$
    - Delete Effects on Success: $\{BoxIn(b, c)\}$
  - $unload(Box : b, Truck : t, City : c)$:
    - Success Probability: if $(BoxOn(b, t) \wedge TruckIn(t, c))$ then .9 else 0
    - Add Effects on Success: $\{BoxIn(b, c)\}$
    - Delete Effects on Success: $\{BoxOn(b, t)\}$
  - $drive(Truck : t, City : c_1, City : c_2)$:
    - Success Probability: if $(TruckIn(t, c_1))$ then 1 else 0
    - Add Effects on Success: $\{TruckIn(t, c_2)\}$
    - Delete Effects on Success: $\{TruckIn(t, c_1)\}$
  - *noop*
    - Success Probability: 1
    - Add Effects on Success: $\emptyset$
    - Delete Effects on Success: $\emptyset$

Symbolic Dynamic Programming, Fig. 1 A formal description of the BoxWorld adapted from Boutilier et al. (2001). We use a simple STRIPS (Fikes and Nilsson 1971) add and delete list representation of actions and, as a simple probabilistic extension in the spirit of PSTRIPS (Kushmerick et al. 1995), we assign probabilities that an action successfully executes conditioned on various state properties

amounts to enumerating all unique configurations of trucks, cities, and boxes. It might then be possible to learn, for example, that taking action *action234* in state *state42* is worth 6. 2 and leads to state *state654321*. Atomic representations are simple, and learning can be implemented using simple lookup tables. These lookup tables, however, can be intractably large as atomic representations easily explode. Furthermore, they do not easily generalize across different numbers of domain objects (We use the term *domain* in the first-order logical sense of an object universe. The term should not be confused with a planning *problem* such as BoxWorld or BlocksWorld.).

In contrast, SDP assumes a relational or first-order logical representation of an MDP (as given in Fig. 1) to exploit the existence of domain objects, relations over these objects, and the ability to express objectives and action effects using quantification.

It is then possible to learn that to get box b to *paris*, the agent drives a truck to the city of b, loads box_1 on the truck, drives the truck to *Paris*, and finally unloads the box box_1 in *Paris*. This is essentially encoded in the symbolic value function shown in Fig. 2, which was computed by discounting rewards t time steps into the future by 0. 9t. The key features to note here are the state and action abstraction in the value and policy representation that are afforded by the first-order specification and solution of the problem. That is, this solution does not refer to any specific set of domain objects, such as $City = \{paris, berlin, london\}$, but rather it provides a solution for *all possible domain ob-*

S

if ($\exists$ *b.BoxIn*(*b*, *paris*)) then do *noop* (value = 100.00)
else if ($\exists$ *b,t.TruckIn*(*t*, *paris*) $\wedge$ *BoxOn*(*b*, *t*)) then do *unload*(*b*, *t*) (value = 89.0)
else if ($\exists$ *b,c,t.BoxOn*(*b*, *t*) $\wedge$ *TruckIn*(*t*, *c*)) then do *drive*(*t*, *c*, *paris*) (value = 80.0)
else if ($\exists$ *b,c,t.BoxIn*(*b*, *c*) $\wedge$ *TruckIn*(*t*, *c*)) then do *load*(*b*, *t*) (value = 72.0)
else if ($\exists$ b, c_1, t, c_2.*BoxIn*(b, c_1) $\wedge$ *TruckIn*(t, c_2)) then do *drive*(t, c_2, c_1) (value = 64.7)
else do *noop* (value = 0.0)

Symbolic Dynamic Programming, Fig. 2 A decision-list representation of the optimal policy and expected discounted reward value for the BoxWorld problem

ject instantiations. And while classical dynamic programming techniques could never solve these problems for large domain instantiations (since they would have to enumerate all states and actions), a domain-independent SDP solution to this particular problem is quite simple due to the power of state and action abstraction.

Background: Markov Decision Processes (MDPs)

In the MDP (Puterman 1994) model, an agent is assumed to fully observe the current state and choose an action to execute from that state. Based on that state and action, nature then chooses a next state according to some fixed probability distribution. In an infinite-horizon MDP, this process repeats itself indefinitely. Assuming there is a reward associated with each state and action, the goal of the agent is to maximize the expected sum of discounted rewards received over an infinite horizon (Although we do not discuss it here, there are other alternatives to discounting such as averaging the reward received over an infinite horizon.). This criterion assumes that a reward received t steps in the future is discounted by γ^t, where γ is a discount factor satisfying $0 \leq \gamma < 1$. The goal of the agent is to choose its actions in order to maximize the expected, discounted future reward in this model.

Formally, a finite state and action MDP is a tuple: $\langle S, A, T, R \rangle$, where S is a finite state space, A is a finite set of actions, T is a transition function: $T : S \times A \times S \rightarrow [0, 1]$, where $T(s, a, \cdot)$ is a probability distribution over S for any $s \in S$ and $a \in A$, and R is a bounded reward function $R : S \times A \rightarrow \mathbb{R}$.

As stated earlier, our goal is to find a policy that maximizes the infinite horizon, discounted reward criterion: $E_\pi[\sum_{t=0}^{\infty} \gamma^t \cdot r_t | s_0]$, where r_t is a reward obtained at time t, γ is a discount factor as defined earlier, π is the policy being executed, and s_0 is the initial starting state. Based on this reward criterion, we define the value function for a policy π as the following:

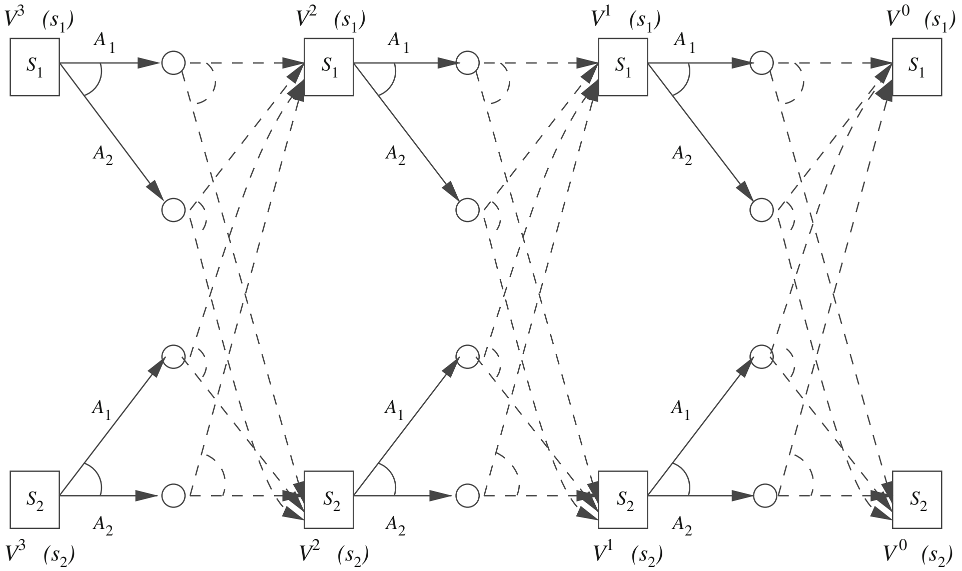
$$V_\pi(s) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t \cdot r_t | s_0 = s\right] \qquad (1)$$

Intuitively, the value function for a policy π is the expected sum of discounted rewards accumulated while executing that policy when starting from state s.

For the MDP model discussed here, the optimal policy can be shown to be stationary (Puterman 1994). Consequently, we use a stationary policy representation of the form $\pi : S \rightarrow A$, with $\pi(s)$ denoting the action to be executed in state s. An optimal policy π^* is the policy that maximizes the value function for all states. We denote the optimal value function over an indefinite horizon as $V^*(s)$ and note that it satisfies the following equality:

$$V^*(s) = \max_{a \in A}\Big\{R(s,a) + \gamma \sum_{s' \in s} T(s,a,s') \cdot V^*(s')\Big\} \qquad (2)$$

Bellman's *principle of optimality* (Bellman 1957) establishes the following relationship between



Symbolic Dynamic Programming, Fig. 3 A diagram demonstrating a *dynamic programming* regression-based evaluation of the MDP value function. *Dashed lines* are used in the expectation computation of the Q-function: for each action, take the expectation over the values of possible successor states. *Solid lines* are used in the max computation: determine the highest valued action to be taken in each state

the optimal value function $V^t(s)$ with a finite horizon of t steps remaining and the optimal value function $V^{t-1}(s)$ with a finite horizon of $t-1$ steps remaining:

$$V^t(s) = \max_{a \in A}\Big\{R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \cdot V^{t-1}(s')\Big\} \qquad (3)$$

A *dynamic programming* approach for computing the optimal value function over an indefinite horizon is known as value iteration and directly implements (3) to compute 1 by successive approximation. As sketched in Fig. 3, we start with arbitrary $V^0(s)$ (e.g., $\forall s V^0(s) = 0$) and perform the Bellman backup given in (3) for each state $V^1(s)$ using the value of $V^0(s)$. We repeat this process for each t to compute $V^t(s)$ from the memorized values for $V^{t-1}(s)$ until we have computed the intended t-stages-to-go value function. $V^t(s)$ will converge to $V^*(s)$ as $t \to \infty$ (Puterman 1994).

Often, the Bellman backup is rewritten in two steps to separate out the action regression and maximization steps. In this case, we first compute the t-stages-to-go Q-function for every action and state:

$$Q^t(s,a) = R(s,a) + \gamma \cdot \sum_{s' \in S} T(s,a,s') \cdot V^{t-1}(s') \qquad (4)$$

Then we maximize over each action to determine the value of the regressed state:

$$V^t(s) = \max_{a \in A}\{Q^t(s,a)\} \qquad (5)$$

This is clearly equivalent to (3) but is in a form that we refer to later, since it separates the algorithm into its two conceptual components: decision-theoretic regression and maximization.

First-Order Markov Decision Processes

A first-order MDP (FOMDP) can be thought of as a universal MDP that abstractly defines the state, action, transition, and reward tuple $\langle S, A, T, R\rangle$ for an infinite number of ground MDPs. To make this idea more concrete, consider the BoxWorld problem defined earlier. While we have not yet formalized the details of the FOMDP representation, it should be clear that the BoxWorld dynamics hold for any instantiation of the domain objects: *Box*, *Truck*, and *City*. For instance, assume that the domain instantiation consists of two boxes $Box = \{box_1, box_2\}$, two trucks $Truck = \{truck_1, truck_2\}$ and two cities $City = \{paris, berlin\}$. Then the resulting ground MDP has 12 state-variable atoms (each atom being *true* or *false* in a state), four atoms for *BoxIn* such as $BoxIn(box_1, paris)$, $BoxIn(box_2, paris)$, ..., four atoms for *TruckIn* such as $TruckIn(truck_2, paris)$, ...and four atoms for *BoxOn* such as $BoxOn(box_2, truck_1)$, ... There are also 24 possible actions (eight for each of *load*,*unload*, and *drive*) such as $load(box_1, truck_1, paris)$, $load(box_1, truck_1, berlin)$, $drive(truck_2, paris, paris)$, $drive(truck_2, paris, berlin)$, etc., where the transition function directly follows from the ground instantions of the corresponding PSTRIPS operators. The reward function looks like: if $(BoxIn(box_1, paris) \vee BoxIn(box\,_2, paris))$ 10 else 0.

Therefore, to solve an FOMDP, we could ground it for a specific domain instantiation to obtain a corresponding ground MDP. Then we could apply classical MDP solution techniques to solve this ground MDP. However, the obvious drawback to this approach is that the number of state variables and actions in the ground MDP grow at least linearly as the domain size increases. And even if the ground MDP could be represented within memory constraints, the number of distinct ground states grows exponentially with the number of state variables, thus rendering solutions that scale with state size intractable even for moderately small numbers of domain objects.

An alternative idea to solving an FOMDP at the ground level is to solve the FOMDP directly at the first-order level using symbolic dynamic programming, thereby obtaining a solution that applies universally to all possible domain instantiations. While the exact representation and SDP solution of FOMDPs differ among the variant formalisms, they all share the same basic first-order representation of rewards, probabilities, and values that we outline next. To highlight this, we introduce a graphical *case notation* to allow first-order specifications of the rewards, probabilities, and values required for FOMDPs:

$$case = \begin{array}{|l|}\hline \phi_1 \ : t_1 \\ \hline : \quad : : \\ \hline \phi_n \ : t_n \\ \hline \end{array}$$

Here the φ_i are *state formulae* and the t_i are terms. Often the t_i are constants and the φ_i partition state space. To make this concrete, we represent our BoxWorld FOMDP reward function as the following *rCase* statement:

$$rCase \ = \begin{array}{|l|}\hline \exists b.BoxIn(b, paris) : 10 \\ \hline \neg\exists b.BoxIn(b, paris) : 0 \\ \hline \end{array}$$

Here we see that the first-order formulae in the case statement divide all possible ground states into two regions of constant value: when there exists a box in Paris, a reward of 10 is achieved, otherwise a reward of 0 is achieved. Likewise, the value function *case* that we derive through SDP can be represented in exactly the same manner. Indeed, as we will see shortly, $case^0 = rCase$ in the first-order version of value iteration.

To state the FOMDP transition function for an action, we decompose stochastic "agent" actions into a *collection* of deterministic actions, each corresponding to a possible outcome of the stochastic action. We then specify a distribution according to which "nature" may choose a deterministic action from this set whenever the stochastic action is executed.

Letting $A(\vec{x})$ be a stochastic action with nature's choices (i.e., deterministic actions) $n_1(\vec{x}), \ldots n_k(\vec{x})$, we represent the distribution over $n_i(\vec{x})$ given $A(\vec{x})$ using the notation $pCase(n_j(\vec{x}), A(\vec{x}))$. Continuing our logistics example, if the success of driving a truck depends on whether the destination city is *paris* (perhaps due to known traffic delays), then we decompose the stochastic *drive* action into two deterministic actions *driveS* and *driveF*, respectively denoting success and failure. Then we can specify a distribution over nature's choice deterministic outcome for this stochastic action:

$$pCase(driveS(t,c_1,c_2)), drive(t,c_1,c_2)) = \begin{array}{|l|}\hline c_2 = paris : 0.6 \\ \hline c_2 \neq paris : 0.9 \\ \hline \end{array}$$

$$pCase(driveF(t,c_1,c_2)), drive(t,c_1,c_2)) = \begin{array}{|l|}\hline c_2 = paris : 0.4 \\ \hline c_2 \neq paris : 0.1 \\ \hline \end{array}$$

Intuitively, to perform an operation on case statements, we simply perform the corresponding operation on the intersection of all case partitions of the operands. Letting each φ_i and ψ_j denote generic first-order formula, we can perform the "cross-sum" $\oplus$ of case statements in the following manner:

$$\begin{array}{|l|}\hline \phi_1 : 10 \\ \hline \phi_2 : 20 \\ \hline \end{array} \oplus \begin{array}{|l|}\hline \psi_1 : 1 \\ \hline \psi_2 : 2 \\ \hline \end{array} = \begin{array}{|l|}\hline \phi_1 \wedge \psi_1 : 11 \\ \hline \phi_1 \wedge \psi_2 : 12 \\ \hline \phi_2 \wedge \psi_1 : 21 \\ \hline \phi_2 \wedge \psi_2 : 22 \\ \hline \end{array}$$

Likewise, we can perform $\ominus$, $\otimes$, and max operations by respectively subtracting, multiplying, or taking the max of partition values (as opposed to adding them) to obtain the result. Some partitions resulting from the application of the $\oplus$, $\ominus$, and $\otimes$ operators may be inconsistent; we simply discard such partitions (since they can obviously never correspond to any world state).

We define another operation on case statements max $\exists\vec{x}$ that is crucial for SDP. Intuitively, the meaning of max $\exists\vec{x}$ $case(\vec{x})$ is a case statement where the maximal value is assigned to each region of state space where there exists a satisfying instantiation of $\vec{x}$. To make these ideas concrete, following is an exposition of how the max $\exists\vec{x}$ may be explicitly computed:

$$\max \exists\vec{x} \begin{array}{|l|}\hline \psi_1(\vec{x}) : 1 \\ \hline \psi_2(\vec{x}) : 2 \\ \hline \psi_3(\vec{x}) : 3 \\ \hline \end{array}$$

$$= \begin{array}{|ll|}\hline \exists\vec{x}\psi_3(\vec{x}) & : 3 \\ \hline \neg(\exists\vec{x}\psi_3(\vec{x})) \wedge \exists\vec{x}\psi_2(\vec{x}) & : 2 \\ \hline \neg(\exists\vec{x}\psi_3(\vec{x})) \wedge \neg(\exists\vec{x}\psi_2(\vec{x})) \wedge \exists\vec{x}\psi_1(\vec{x}) & : 1 \\ \hline \end{array}$$

Here we have simply sorted partitions in order of values and have ensured that the highest value is assigned to partitions in which there exists a satisfying instantiation of $\vec{x}$ by rendering lower value partitions disjoint from their higher-value antecedents.

Symbolic Dynamic Programming

SDP is a dynamic programming solution to FOMDPs that produces a logical case description of the optimal value function. This is achieved through the operations of first-order decision-theoretic regression (FODTR) and symbolic maximization that perform the traditional dynamic programming Bellman backup at an abstract level without explicit enumeration of either the state or action spaces of the FOMDP. Among many uses, the application of SDP leads to a domain-independent value iteration solution to FOMDPs.

Suppose that we are given a value function in the form *case*. The FODTR (Boutilier et al. 2001) of this value function through an action $A(\vec{x})$ yields a case statement containing the logical description of states and values that would give rise to *case* after doing action $A(\vec{x})$. This is analogous to classical goal regression, the key difference being that action $A(\vec{x})$ is stochastic. In MDP terms, the result of FODTR is a case statement representing a Q-function.

We define the *FODTR* operator in the following manner:

$$FODTR[vcase, A(\vec{x})] = rcase \oplus \gamma[\oplus_j\{pCase(n_j(\vec{x})) \otimes \; Regr[vcase, A(\vec{x})]\}] \quad (6)$$

Note that we have not yet defined the regression operator $Regr[vcase, A(\vec{x})]$. As it turns out, the implementation of this operator is specific to a given FOMDP language and SDP implementation. We simply remark that the regression of a formula ψ through an action $A(\vec{x})$ is a formula ψ' that holds prior to $A(\vec{x})$ being performed iff ψ holds after $A(\vec{x})$. However, regression is a deterministic operator and thus FODTR takes the expectation of the regression over all possible outcomes of a stochastic action according to their respective probabilities.

It is important to note that the case statement resulting from FODTR contains free variables for the action parameters $\vec{x}$. That is, for any constant binding $\vec{c}$ of these action parameters such that $\vec{x} = \vec{c}$, the case statement specifies a well-defined logical description of the value that can be obtained by taking action $A(\vec{c})$ and following a policy so as to obtain the value given by *v*{ *case* thereafter. However, what we really need for symbolic dynamic programming is a logical description of a Q-function that tells us the highest value that can be achieved for *any* action instantiation. This leads us to the following $qCase(A(\vec{x}))$ definition of a first-order Q-function that makes use of the previously defined $\exists\vec{x}$ operator:

$$qCase^t(A(\vec{x})) \\ = \max \exists\vec{x}.FODTR[vcase^{t-1}, A(\vec{x})] \quad (7)$$

Intuitively, $qCase^t(A(\vec{x}))$ is a logical description of the Q-function for action $A(\vec{x})$ indicating the best value that could be achieved by *any* instantiation of $A(\vec{x})$. And by using the case representation and action quantification in the $\max \exists\vec{x}$ operation, FODTR effectively achieves *both* action and state abstraction.

At this point, we can regress the value function through a *single* action, but to complete the dynamic programming step, we need to know the maximum value that can be achieved by *any* action (e.g., in the BoxWorld FOMDP, our possible action choices are *unload*(b, t, c), *load*(b, t, c), and *drive*(t, c_1, c_2)). Fortunately, this turns out to be quite easy. Assuming we have m actions $\{A_1(\vec{x}_1), \ldots, A_m(\vec{x}_m)\}$, we can complete the SDP step in the following manner using the previously defined max operator:

$$vcase^t = \max_{a \in \{A_1(\vec{x}_1),\ldots,A_m(\vec{x}_m)\}} qCase^t(a) \quad (8)$$

While the details of SDP may seem very abstract at the moment, there are several examples for specific FOMDP languages that implement SDP as described earlier, for which we provide references next. Nonetheless, one should note that the SDP equations given here are exactly the “lifted” versions of the traditional dynamic programming solution to MDPs given previously in (4) and (5). The reader may verify — on a conceptual level — that applying SDP to the 0-stages-to-go value function (i.e., *case*^{0} = *rCase*, given previously) yields the following 1- and 2-stages-to-go value functions in the BoxWorld domain (¬ “indicating the conjunction of the negation of all higher value partitions):

$$case^t = \begin{array}{|lr|}\hline \exists b.BoxIn(b, paris) : & 19.0 \\ \hline \neg\text{“} \wedge \exists b,t.TruckIn(t, paris) \wedge BoxOn(b,t) : & 9.0 \\ \hline \neg'' : & 0.0 \\ \hline \end{array}$$

$$case^2 = \begin{array}{|lr|}\hline \exists b.BoxIn(b, paris) : & 27.1 \\ \hline \neg\text{“} \wedge \exists b,t.TruckIn(t, paris) \wedge BoxOn(b,t) : & 17.1 \\ \hline \neg\text{“} \wedge \exists b,c.BoxOn(b,t) \wedge TruckIn(t,c) : & 8.1 \\ \hline \neg'' : & 0.0 \\ \hline \end{array}$$

After sufficient iterations of SDP, the t-stages-to-go value function converges, giving the optimal value function (and corresponding policy) from Fig. 2.

Applications

Variants of SDP have been successfully applied in decision-theoretic planning domains such as BLOCKSWORLD, BOXWORLD, ZENOWORLD, ELEVATORS, DRIVE, PITCHCATCH, and SCHEDULE. The first-order approximate linear programming (FOALP) system (Sanner and Boutilier 2005) was runner-up at the probabilistic track of the 5th International Planning Competition (IPC-6). Related techniques have been used to solve path planning problems within robotics and instances of real-time strategy games, Tetris, and Digger.

Future Directions

The original *SDP* (Boutilier et al. 2001) approach is a value iteration algorithm for solving FOMDPs based on Reiter's situations calculus. Since then, a variety of exact algorithms have been introduced to solve MDPs with relational (RMDP) and first-order (FOMDP) structure (We use the term *relational MDP* to refer to models that allow implicit existential quantification, and *FOMDP* for those with explicit existential and universal quantification.). *First-order value iteration (FOVIA)* (Hölldobler and Skvortsova 2004; Karabaev and Skvortsova 2005) and the *relational Bellman algorithm (ReBel)* (Kersting et al. 2004) are value iteration algorithms for solving RMDPs. In addition, *first-order decision diagrams (FODDs)* have been introduced to compactly represent case statements and to permit efficient application of SDP operations to solve RMDPs via value iteration (Wang et al. 2007) and policy iteration (Wang and Khardon 2007). All of these algorithms have some form of guarantee on convergence to the (ε-)optimal value function or policy. The expressiveness of FOMDPs has been extended to support indefinitely factored reward and transition functions in FOMDPs (Sanner and Boutilier 2007).

A class of linear-value approximation algorithms have been introduced to approximate the value function as a linear combination of weighted basis functions. FOALP (Sanner and Boutilier 2005) directly approximates the FOMDP value function using a linear program. *First-order approximate policy iteration (FOAPI)* (Sanner and Boutilier 2006) approximately solves for the FOMDP value function by iterating between policy and value updates in a policy-iteration style algorithm. Somewhat weak error bounds can be derived for a value function approximated via FOALP (Sanner and Boutilier 2005) while generally stronger bounds can be derived from the FOAPI solution (Sanner and Boutilier 2006).

Finally, there are a number of heuristic solutions to FOMDPs and RMDPs. *Approximate policy iteration* (Fern et al. 2003) induces rule-based policies from sampled experience in small-domain instantiations of RMDPs and generalizes these policies to larger domains. In a similar vein, *inductive policy selection using first-order regression* (Gretton and Thiebaux 2004) uses the action regression operator in the situation calculus to provide the first-order hypothesis space for an inductive policy learning algorithm. *Approximate linear programming (for RMDPs)* (Guestrin et al. 2003) is an approximation technique using linear program optimization to find a best-fit value function over a number of sampled RMDP domain instantiations.

Cross-References

- ▶ Dynamic Programming
- ▶ Markov Decision Processes

Recommended Reading

Bellman RE (1957) Dynamic programming. Princeton University Press, Princeton

Boutilier C, Reiter R, Price B (2001) Symbolic dynamic programming for first-order MDPs. In: IJCAI-01, Seattle, pp 690–697

Fern A, Yoon S, Givan R (2003) Approximate policy iteration with a policy language bias. In: NIPS-2003, Vancouver

Fikes RE, Nilsson NJ (1971) STRIPS: a new approach to the application of theorem proving to problem solving. Artif Intell 2:189–208

Gretton C, Thiebaux S (2004) Exploiting first-order regression in inductive policy selection. In: UAI-04, Banff, pp 217–225

Guestrin C, Koller D, Gearhart C, Kanodia N (2003) Generalizing plans to new environments in relational MDPs. In: IJCAI-03, Acapulco

Hölldobler S, Skvortsova O (2004) A logic-based approach to dynamic programming. In: AAAI-04 workshop on learning and planning in MDPs, Menlo Park, pp 31–36

Karabaev E, Skvortsova O (2005) A heuristic search algorithm for solving first-order MDPs. In: UAI-2005, Edinburgh, pp 292–299

Kersting K, van Otterlo M, De Raedt L (2004) Bellman goes relational. In: ICML-04. ACM Press, New York

Kushmerick N, Hanks S, Weld D (1995) An algorithm for probabilistic planning. Artif Intell 76:239–286

Puterman ML (1994) Markov decision processes: discrete stochastic dynamic programming. Wiley, New York

Sanner S, Boutilier C (2005) Approximate linear programming for first-order MDPs. In: UAI-2005, Edinburgh

Sanner S, Boutilier C (2006) Practical linear evaluation techniques for first-order MDPs. In: UAI-2006, Boston

Sanner S, Boutilier C (2007) Approximate solution techniques for factored first-order MDPs. In: ICAPS-07, Providence, pp 288–295

Wang C, Khardon R (2007) Policy iteration for relational MDPs. In: UAI, Vancouver

Wang C, Joshi S, Khardon R (2007) First order decision diagrams for relational MDPs. In: IJCAI, Hyderabad

Symbolic Regression

▸ Equation Discovery

Symmetrization Lemma

Synonyms

Basic lemma

Definition

Given a distribution P over a sample space $\mathcal{Z}$, a finite sample $\mathbf{z} = (z_1, \ldots, z_n)$ drawn i.i.d. from P and a function $f : \mathcal{Z} \to \mathbb{R}$ we define the shorthand $\mathbb{E}_P f = \mathbb{E}_P[f(z)]$ and $\mathbb{E}_{\mathbf{z}} f = \frac{1}{n} \sum_{i-1}^{n} f(z_i)$ to denote the true and empirical average of f. The symmetrization lemma is an important result in the learning theory as it allows the true average $\mathbb{E}_P f$ found in ▸ generalization bounds to be replaced by a second empirical average $\mathbb{E}_{\mathbf{z}'} f$ taken over an independent *ghost sample* $\mathbf{z}' = z'_1, \ldots z'_n$ drawn i.i.d. from P. Specifically, the symmetrization lemma states that for any $\epsilon > 0$ whenever $n\epsilon^2 \geq 2$

$$P^n \left(\sup_{f \in \mathbb{F}} |\mathbb{E}_P f - \mathbb{E}_{\mathbf{z}} f| > \epsilon \right) \leq 2P^{2n} \left(\sup_{f \in \mathbb{F}} |\mathbb{E}_{\mathbf{z}'} f - \mathbb{E}_{\mathbf{z}} f| > \frac{\epsilon}{2} \right). \quad (1)$$

This means the typically difficult to analyze behavior of $\mathbb{E}_P f$ – which involves the entire sample space $\mathcal{Z}$ – can be replaced by the evaluation of functions from $\mathbb{F}$ over the points in $\mathbf{z}$ and $\mathbf{z}'$.

Synaptic Efficacy

▸ Weight

T

Table Extraction

► Table Extraction from Text Documents

Table Extraction from Text Documents

James Hodson
AI for Good Foundation,
New York, NY, USA

Abstract

Tables appear in text documents in almost every form imaginable, from simple lists to nested, hierarchical, and multidimensional layouts. They are primarily designed for human consumption and therefore can require a wide variety of visual cues and interpretive capabilities to be fully understood. This chapter deals with the challenges machines face when attempting to process and understand tables, along with state-of-the-art methods and performance on this task.

Synonyms

Table extraction; Table parsing; Table understanding

Definition

The objective of table extraction is to convert human-focused notation, to a logical, machine-readable, and machine-understandable form. This task is closely related to and could be viewed as a subproblem of document structure extraction. It is generally considered a higher level natural language processing problem, requiring a pipeline of capabilities to address.

Motivation and Background

Tables appear in text documents in almost every form imaginable, from simple lists to nested, hierarchical, and multidimensional layouts. They are primarily designed for human consumption and therefore can require a wide variety of visual cues and interpretive capabilities to be fully understood. In fact, the assumption of human consumption allows for a breadth of content presentation that is practically limitless. Often, critical information that is relevant to the interpretation is assumed, provided in short-hand notation, or inferred from other aspects of the content or layout.

Given that the problem of table extraction is motivated by the presence of electronic documents, it has only been formally studied since the early 1990s, as the prevalence of computer-based document storage, editing, and retrieval increased (Laurentini et al. 1992; Guthrie et al. 1993). With hundreds of document formats,

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

layout preferences, and established customs for data interchange, the problem has only become worse at web-scale, with very few document originators choosing machine-readable syntax over visual layouts (i.e., drawing).

This article explores the genesis of the problem domain, how to formalize and break down the various tasks involved in building a table extraction solution, and the methodologies generally used.

There have been several attempts at formalizing the table model and notation. Some of these were designed independently of automatic table extraction research (Association of American Publishers 1986) and pertain to the best practices for tabular data design. Computationally driven table models generally refer to the widely used Augmented Wang Notation (Wang 1996) which specifies a hierarchical schema for describing types, classes, and relations among cells. Common table models are necessary for the interoperation of different stages of the extraction pipeline as well as the common evaluation of different approaches with the same gold standard reference data (Govindaraju et al. 2013). As in most machine learning pipelines, it is often convenient to isolate component parts for algorithmic development and testing.

Approaches to evaluation of table extraction techniques vary widely, and can be looked at from multiple perspectives document level, table level, access level, and cell level. Each stage of the extraction pipeline can be evaluated separately, or one can look at the overall goal achievement measures. Vanessa Long (2010) adopts a multilevel structural evaluation approach which can be particularly informative.

Recent work is part of a more sparse literature, with consistently decreased focus since the early 2000s. In spite of this, table extraction is not a problem that has any broadly adopted solution. It is a fragmented environment and often viewed as a practitioner's problem as part of larger systems. However, certain industries (e.g., finance) and the rise of web-scale information extraction have led to a renewed focus on these technologies in a research and applied setting (Mitchell et al. 2015).

Structure of the Learning System

We will consider each of the logical steps that form part of a complete table extraction system. Hurst (2000) and Fang et al. (2012) both propose pipelines that allow for the evaluation of discrete capabilities. Starting from a raw text document, each subsequent pass adds more and more structure, getting us closer to the final goal – a disambiguated relational table object. Approaches at each stage can consider not only textual features but also layout and other visual cues. In fact, it is often the case that table extraction techniques on text documents will use a variety of methodologies from the computer vision community.

Table Detection

Given a text document, the objective is to identify whether or not it contains a table object. It should be possible to signal when a document contains multiple such distinct objects and their rough contiguous location (Kornfeld and Wattecamps 1998). Often, this step is combined with the next (boundary detection) to perform joint detection and delineation of tabular areas.

In the case where detection is performed in isolation, it may be viewed as a binary classification, sequence labeling, or clustering task over the document. Lopresti et al. (2000) approached this problem from a text density/clustering perspective over single-columned ASCII text documents, though more recent efforts in industrial applications tend to benefit from cascading binary (SVM) classifiers or random forest approaches.

Table Boundary Identification

Table boundary identification recognizes the boundaries of detected tables such that they could be isolated from the surrounding information. Laurentini et al. (1992) makes use of the Hough transform to identify connected shapes and components that represent the margins of tables. These must be separated from charts, images, and other visual components, which is the aim of the table detection step mentioned above. The identification of table boundaries can also benefit the table detection task by providing additional structural features on which to predicate the distinction from other visual objects in a document.

Structural Inference

For each recognized table, identify the column and row structure, such that each cell could be uniquely identified. In practice, the methods applied to this task mirror those of table boundary identification. However, there are additional constraints that often make it worthwhile to consider this step separately. For example, tables are structurally constrained to maintain linear relationships among cells – rows and columns must remain broadly coherent. Furthermore, the task may be recursive, where tables contain tables, or other structural items as inserts. It is important that this step provide the most accurate microstructure possible. As such, it can often be beneficial to look at measures of content coherence for merging or splitting neighbors, at the same time as optimizing overall coherence.

Functional Classification

The logical definition of a table is that of a set of associated keys and values. Headers, or groupings of headers, represent keys, which intersect along the axes of a table. The intersects of these header cells represent the values of interest. Headers provide the information necessary to understand the type of data as well as uniquely pinpoint the location of each value. Functional classification, therefore, identifies for each cell, whether it represents a key or a value (Liu 2009).

Functional Interpretation

For each cell representing a value, classify its type (e.g., weight, location, distance, revenue), according to its associated headers. In addition, many tables rely on auxiliary information and interpretation, such as footnotes or implied coherence (e.g., all adjacent cells have the same property, but do not explicitly define it). These additional structures need to be identified and associated with each cell. Furthermore, cell values should be fully normalized according to the available information. If a header states that all values are in $M, all numbers should take this into account.

Disambiguation

In most cases, the reason for reading and extracting a table from text is to be able to work with the information held therein. Comparing values to prior years' numbers, reasoning about them, and filtering all require that the data fit into some logical representation of the domain of interest, whether implicitly or explicitly defined. Disambiguating the values allows them to be used consistently and stored uniformly for later querying (Liu 2009; Hurst 2000).

Disambiguation requires some desired final representation, whether a formal ontology or a relational database schema. Ideally the representation would cover the entire universe of interest, allowing every possible value type to be logically captured. However, it is often necessary to account for content that has not been encountered before.

Generally, disambiguation can be viewed as a supervised classification problem, whereby explicit or implicit (latent, structural) features are mapped probabilistically to available outcomes, constrained by meta-schemas such as length, primitive type, and relative position. Additionally, structural factors (number of values, etc.) can be used, within an iterative framework, to further limit the output space. That is, as more of the table has been disambiguated, fewer options remain that would be consistent with the prior results. As such, this can be viewed as a constrained optimization, where the schema is sufficiently well defined.

Cross-References

- ▶ Semantic Annotation of Text Using Open Semantic Resources
- ▶ Entity Resolution

T

Recommended Reading

Association Of American Publishers (1986) Markup of tabular material. Technical report. Association of American Publishers, Manuscript Series

Fang, J, Mitra P, Tang Z, Lee GC (2012) Table header detection and classification. In: Proceedings of AAAI, Toronto

Göbel M, Hassan T, Oro E, Orsi G (2012) A methodology for evaluating algorithms for table understanding in PDF documents. In: Proceedings of the 2012 ACM symposium on document engineering, Atlanta, pp 45–48

Govindaraju V, Zhang C, Ré C (2013) Understanding tables in context using standard NLP toolkits. In: Proceedings of the ACL, Sofia

Guthrie J, Weber S, Kimmerly N (1993) Searching documents: cognitive processes and deficits in understanding graphs, tables, and illustrations. Contemp Educ Psychol 18:186–221

Hurst MF (2000) The interpretation of tables in texts. Ph.D. thesis, University of Edinburgh, Edinburgh

Kornfeld W, Wattecamps J (1998) Automatically locating, extracting and analyzing tabular data. In: SIGIR'98: Proceedings of the 21st annual international ACM SIGIR conference, Melbourne, pp 347–348

Laurentini A, Viada P (1992) Identifying and understanding tabular material in compound documents. In: Proceedings of 11th IAPR international conference on pattern recognition. Conference B: pattern recognition methodology and systems, IEEE, The Hague, vol II, pp 405–409

Liu Y (2009) Tableseer: automatic table extraction, search, and understanding. Ph.D. thesis, The Pennsylvania State University

Long V (2010) An agent-based approach to table recognition and interpretation. Ph.D. thesis, Macquarie University, Sydney

Lopresti D, Hu J, Kashi R, Wilfong G (2000) Medium-independent table detection. In: SPIE document recognition and retrieval VII, San Jose, pp 291–302

Mitchell T, Cohen W, Hruschka E, Talukdar P, Betteridge J, Carlson A, Dalvi B, Gardner M, Kisiel B, Krishnamurthy J, Lao N, Mazaitis K, Mohamed T, Nakashole N, Platanios E, Ritter A, Samadi M, Settles B, Wang R, Wijaya D, Gupta A, Chen X, Saparov A, Greaves M, Welling J (2015) In Proceedings of the Conference on Artificial Intelligence (AAAI)

Padmanabhan R, Jandhyala RC, Krishnamoorthy M, Nagy G, Seth S, Silversmith W (2009) Interactive conversion of Large web tables. In: Proceedings of eighth international workshop on graphics recognition, GREC 2009. City University of La Rochelle, La Rochelle

Sarawagi S, Chakrabarti S (2014) Open-domain quantity queries on Web tables: annotation, response, and consensus models. In: Proceedings of ACM SIGKDD, New York

Shafait F, Smith R (2010) Table detection in heterogeneous documents. In: Proceedings of the 9th IAPR international workshop on document analysis systems, Boston, pp 65–72

Thompson M (1996) A tables manifesto. In: Proceedings of SGMK Europe, Munich, pp 151–153

Wang X (1996) Tabular abstraction, editing, and formatting. Ph.D. thesis, University of Waterloo, Waterloo

Table Parsing

▶ Table Extraction from Text Documents

Table Understanding

▶ Table Extraction from Text Documents

Tagging

▶ POS Tagging

TAN

▶ Tree Augmented Naive Bayes

Taxicab Norm Distance

▶ Manhattan Distance

TD-Gammon

Definition

TD-Gammon is a world-champion strength backgammon program developed by Gerald Tesauro. Its development relied heavily on machine learning techniques, in particular on ▶ Temporal-Difference Learning. Contrary to successful game programs in domains such as chess, which can easily out-search their human opponents but still trail these ability of estimating the positional merits of the current board configuration, TD-GAMMON was able to excel in backgammon for the same reasons that humans play well: its grasp of the positional strengths and weaknesses was excellent. In 1998, it lost a 100-game competition against the world champion with only 8 points. Its sometimes unconventional but very solid evaluation of certain opening strategies had a strong impact on the backgammon community and was soon adapted by professional players.

Description of the Learning System

TD-Gammon is a conventional game-playing program that uses very shallow search (the first versions only searched one ply) for determining its move. Candidate moves are evaluated by a ▸ Neural Network, which is trained by TD(λ), a well-known algorithm for Temporal-Difference Learning (Tesauro 1992). This evaluation function is trained on millions of games that the program played against itself. At the end of each game, a reinforcement signal that indicates whether the game has been lost or won is passed through all moves of the game. TD-Gammon developed useful concepts in the hidden layer of its network. Tesauro (1992) shows examples for two hidden units of TD-Gammon that he interpreted as a race-oriented feature detector and an attack-oriented feature detector.

TD-Gammon clearly surpassed its predecessors, in particular the Computer Olympiad champion Neurogammon, which was trained with ▸ Preference Learning (Tesauro 1989). In fact, early versions of TD-Gammon, which only used the raw board information as features, already learned to play as well as Neurogammon, which used a sophisticated set of features. Adding more sophisticated features to the input representation further improved TD-Gammon's playing strength. Over time, TD-Gammon not only that increase the number of training games that it played against itself, but Tesauro also increased the search depth and changed the network architecture, so that TD-Gammon eventually reached world-championship strength (Tesauro 1995).

Cross-References

▸ Machine Learning and Game Playing

Recommended Reading

Tesauro G (1989) Connectionist learning of expert preferences by comparison training. In: Touretzky D (ed) Proceedings of the advances in neural information processing systems 1 (NIPS-88). Morgan Kaufmann, San Francisco, pp 99–106

Tesauro G (1992) Practical issues in temporal difference learning. Mach Learn 8:257–278. http://mlis.www.wkap.nl/mach/abstracts/absv8p257.htm

Tesauro G (1995) Temporal difference learning and TD-Gammon. Commun ACM 38(3):58–68. http://www.research.ibm.com/massdist/tdl.html

TDIDT Strategy

▸ Divide-and-Conquer Learning

Temporal Credit Assignment

▸ Credit Assignment

Temporal Data

▸ Time Series

Temporal Difference Learning

William Uther
NICTA and The University of New South Wales,
Sydney, NSW, Australia

Definition

Temporal Difference Learning, also known as TD-Learning, is a method for computing the long term utility of a pattern of behavior from a series of intermediate rewards (Sutton 1984, 1988, 1998). It uses differences between successive utility estimates as a feedback signal for learning. The Temporal Differencing approach to model-free ▸ reinforcement learning was introduced by, and is often associated with, R.S. Sutton. It has ties to both the artificial intelligence and psychological theories of reinforcement learning as well as ▸ dynamic programming and operations research from economics (Bellman 1957; Samuel 1959; Watkins 1989; Puterman 1994; Bertsekas 1996).

While TD learning can be formalised using the theory of ▸ Markov Decision Processes, in many cases it has been used more as a heuristic technique and has achieved impressive results even in situations where the formal theory does not strictly apply, e.g., Tesauro's TD-Gammon (Tesauro 1995) achieved world champion performance in the game of backgammon. These heuristic results often did not transfer to other domains, but over time the theory behind TD learning has expanded to cover large areas of reinforcement learning.]

Formal Definitions

Consider an agent moving through a world in discrete time steps, $t_1,\ t_2, \ldots$. At each time step, t, the agent is informed of both the current state of the world, $s_t \in \mathcal{S}$, and its reward, or utility, for the previous time step, $r_{t-1} \in \mathbb{R}$.

As the expected long term utility of a pattern of behavior can change depending upon the state, the utility is a function of the state, V:$\mathcal{S} \to \mathbb{R}$. V is known as the *value function* or *state-value function*. The phrase "long term utility" can be formalized in multiple ways.

Undiscounted Sum of Reward

The simplest definition is that long term reward is the sum of all future rewards.

$$\begin{aligned} V(s_t) &= r_t + r_{t+1} + r_{t+2} + \ldots \\ &= \sum_{\delta=0}^{\infty} r_t + \delta \end{aligned}$$

Unfortunately, the undiscounted sum of reward is only well defined if this sum converges. Convergence is usually achieved by the addition of a constraint that the agent's experience terminates at some, finite, point in time and all rewards after that point are zero.

Discounted Sum of Reward

The *discounted* utility measure discounts rewards exponentially into the future.

$$\begin{aligned} V(s_t) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \quad \gamma \in [0,1] \\ &= \sum_{\delta=0}^{\infty} r_{t+\delta} \end{aligned}$$

Note that when $\gamma = 1$ the discounted and undiscounted regimes are identical. When $\gamma < 1$, the discounted reward scheme does not require that the agent experience terminates at some finite time for convergence. The *discount factor* γ can be interpreted as an inflation rate, a probability of failure for each time step, or simply as a mathematical trick to achieve convergence.

Average Reward

Rather than consider a sum of rewards, the *average reward* measure of utility estimates both the expected reward per future time step, also known as the *gain*, and the current difference from that long-term average, or *bias*.

$$G(s_t) = \lim_{n\to\infty} \frac{1}{n} \sum_{\delta=0}^{n} r_{t+\delta}$$

$$B(s_t) = \sum_{\delta=0}^{\infty} [r_{t+\delta} - G(s_{t+\delta})]$$

A system where any state has a nonzero probability of being reached from any other state is known as an ergodic system. For such a system the gain, $G(s)$, will have the same value for all states and the bias, $B(s)$, serves a similar purpose to $V(s)$ above in indicating the relative worth of different states. While average reward has a theoretical advantage in that there is no discount factor to choose, historically average reward has been considered more complex to use than the discounted reward regimes and so has been less used in practice. There is a strong theoretical relationship between average reward and discounted reward in the limit as the discount factor approaches one.

Here we focus on discounted reward.

Estimating Discounted Sum of Reward

The temporal differencing estimation procedure is based on recursive reformulation of the above definitions. For the discounted case:

$$\begin{aligned} V(s_t) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots \\ &= r_t + \gamma[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ldots] \\ &= r_t + \gamma V(s_{t+1}) \end{aligned}$$

From the recursive formulation we can see that the long term utility for one time step is closely related to the long term utility at the next time step. If there is already an estimate of the long term utility at s_t, $V(s_t)$, then we could calculate a change in that value given a new trajectory as follows:

$$\Delta_t = [r_t + \gamma V(s_{t+1})] - V(s_t)$$

If we are dealing with a stochastic system, then we may not want to update $V(s_t)$ to the new value in one jump, but rather only move part way toward the new value:

$$\Delta_t = \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$$

where α is a learning rate between 0 and 1. As an assignment, this update can be written in a number of equivalent ways, the two most common being:

$$V(s_t) \leftarrow V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t)) \ or,$$

$$V(s_t) \rightarrow (1-\alpha)V(s_t) + \alpha(r_t + \gamma V(s_{t+1}))$$

This *update*, *error*, *learning* or *delta* rule is the core of temporal difference learning. It is from this formulation, which computes a delta based on the difference in estimated long term utility of the world at two consecutive time steps, that we get the term temporal differencing.

Having derived this update rule, we can now apply it to finding the long term utility of a particular agent. In the simplest case we will assume that there are a finite number of Markov states of the world, and that these can be reliably detected by the agent at run time. We will store the function V as an array of real numbers, with one number for each world state.

After each time step, t, we will use the knowledge of the previous state, s_t, the instantaneous reward for the time step, r_t, and the resulting state, s_{t+1}, to update the value of the previous state, $V(s_t)$, using the delta rule above:

$$V(s_t) \leftarrow V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$$

Eligibility Traces and TD (λ)

Basic temporal differencing as represented above can be quite slow to converge in many situations. Consider, for example, a simple corridor with a single reward at the end, and an agent that walks down the corridor. Assume that the value function was initialized to a uniform zero value. On each walk down the corridor, useful information is only pushed one step back toward the start of the corridor.

Eligibility traces try to alleviate this problem by pushing information further back along the trajectory of the agent with each update to V. An algorithm incorporating eligibility traces can be seen as a mixture of "pure" TD, as described above, and ▶ Monte-Carlo estimation of the long term utility. In particular, the λ parameter to the TD(λ) family of algorithms specifies where in the range from pure TD, when $\lambda = 0$, to pure Monte-Carlo, when $\lambda = 1$, a particular algorithm falls.

Eligibility traces are implemented by keeping a second function of the state space , $\epsilon : \mathcal{S} \rightarrow \mathbb{R}$. The ε function represents how much an experience now should affect the value of a state the agent has previously passed through. When the agent performs an update, the values of all states are changed according to their eligibility.

The standard definition of the eligibility of a particular state uses an exponential decay over time, but this is not a strict requirement and other definitions of eligibility could be used. In addition, each time a state is visited, its eligibility increases. Formally, on each time step,

$$\forall_{s \in \mathcal{S}} \varepsilon(s) \leftarrow \gamma\lambda\varepsilon(s) \quad \text{and then,}$$
$$\varepsilon(s_t) \leftarrow \varepsilon(s_t) + 1$$

This eligibility is used to update all state values by first calculating the delta for the current state as above, but then applying it to all states according to the eligibility values:

$$\delta_t = \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$$
$$\forall_{s \in \mathcal{S}} V(s) \leftarrow V(s) + \delta_t \varepsilon(s)$$

Convergence

TD value function estimation has been shown to converge under many conditions, but there are also well known examples where it does not converge at all, or does not converge to the correct long term reward (Tsitsiklis 1997).

In particular, temporal differencing has been shown to converge to the correct value of the long term discounted reward if,

- The world is finite.
- The world state representation is Markovian.
- The rewards are bounded.
- The representation of the V function has no constraints (e.g., a tabular representation with an entry for each state).
- The learning rate, α, is reduced according to the Robbins-Monro conditions: $\sum_{t=0}^{\infty} \alpha_t = \infty$, and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$.

Much of the further work in TD learning since its invention has been in finding algorithms that provably converge in more general cases.

These convergence results require that a Markovian representation of state be available to the agent. There has been research into how to acquire such a representation from a sequence of observations. The approach of the Temporal Differencing community has been to use TD-Networks (Sutton 2004).

Control of Systems

Temporal Difference Learning is used to estimate the long term reward of a pattern of behavior. This estimation of utility can then be used to improve that behavior, allowing TD to help solve a reinforcement learning problem. There are two common ways to achieve this: An *Actor-Critic* setup uses value function estimation as one component of a larger system, and the *Q-learning* and *SARSA* techniques can be viewed as slight modifications of the TD method which allow the extraction of control information more directly from the value function.

First we will formalise the concept of a pattern of behavior. In the preceding text it was left deliberately vague as TD can be applied to multiple definitions. Here we will focus on discrete action spaces.

Assume there is a set of allowed actions for the agent, $\mathcal{A}$. We define a *Markov policy* as a function from world states to actions, $\pi : \mathcal{S} \rightarrow \mathcal{A}$. We also define a *stochastic* or *mixed* Markov policy as a function from world states to probability distributions over actions, $\pi : \mathcal{S} \rightarrow \mathcal{A} \rightarrow [0, 1]$. The goal of the control algorithm is to find an optimal policy: a policy that maximises long term reward in each state. (When function approximation is used (see section "Approximation"), this definition of an optimal policy no longer suffices. One can then either move to average reward if the system is ergodic, or give a, possibly implicit, weighting function specifying the relative importance of different states.)

Actor-Critic Control Systems

Actor-Critic control is closely related to *mixed policy iteration* from Markov Decision Process theory. There are two parts to an actor-critic system; the *actor* holds the current policy for the agent, and the *critic* evaluates the actor and suggests improvements to the current policy.

There are a number of approaches that fall under this model. One early approach stores a preference value for each world state and action pair, $p : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. The actor then uses a stochastic policy based on the Gibbs softmax function applied to the preferences:

$$\pi(s,a) = \frac{e^{p(s,a)}}{\sum_{x \in \mathcal{A}} e^{P(s,x)}}$$

The critic then uses TD to estimate the long term utility of the current policy, and also uses the TD update to change the preference values. When the agent is positively surprised it increases the preference for an action, when negatively surprised it decreases the preference for an action. The size of the increase or decrease is modulated by a parameter, β:

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t$$

Convergence of this algorithm to an optimal policy is not guaranteed.

A second approach requires the agent to have an accurate model of its environment. In this approach the critic uses TD to learn a value function for the current behavior. The actor uses model based forward search to choose an action likely to lead to a state with a high expected long term utility. This approach is common in two player, zero sum, alternating move games such as Chess or Checkers where the forward search is a deterministic game tree search.

More modern approaches which guarantee convergence are related to *policy gradient* approaches to reinforcement learning (Di 2010). These store a stochastic policy in addition to the value function, and then use the TD updates to estimate the gradient of the long term utility with respect to that policy. This allows the critic to adjust the policy in the direction of the negative gradient with respect to long term value, and thus improve the policy.

Other Value Functions

The second class of approaches to using TD for control relies upon extending the value function to estimate the value of multiple actions. Instead of V we use a *state-action value function*, $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. The update rule for this function is minimally modified from the TD update defined for V above.

Once these state-action value functions have been estimated, a policy can be selected by choosing for each state the action that maximizes the state-action value function, and then adding some exploration.

In order for this extended value function to be learned, the agent must explore each action in each state infinitely often. Traditionally this has been assured by making the agent select random actions occasionally, even when the agent believes that action would be sub-optimal. In general the choice of when to explore using a sub-optimal action, the *exploration/exploitation trade-off*, is difficult to optimize. More recent approaches to optimizing the exploration/exploitation trade-off in reinforcement learning estimate the variance of the value function to decide where they need to explore (Auer 2007).

The requirement for exploration leads to two different value functions that could be estimated. The agent could estimate the value function of the pattern of behavior currently being executed, which includes the exploration. Or, the agent could estimate the value function of the current best policy, excluding the exploration currently in use. These are referred to as *on-policy* and *off-policy* methods respectively.

Q-Learning is an off-policy update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma V(s_{t+1}) - Q(s_t, a_t))$$

Where $V(s_{t+1}) = \max_{a \in \mathcal{A}} Q(s_{t+1}, a)$

SARSA is an on-policy update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

Then for both:

$$\pi(s) = argmax_{a \in \mathcal{A}} Q(s, a)$$

and some exploration.

As can be seen above, the update rules for SARSA and Q-learning are very similar – they only differ in the value used for the resulting state. Q-learning uses the value of the best action, whereas SARSA uses the value of the action that will actually be chosen.

Q-Learning converges to the best policy to use once you have converged and can stop exploring. SARSA converges to the best policy to use if you want to keep exploring as you follow the policy (Lagoudakis 2003).

Approximation

A major problem with many state based algorithms, including TD learning, is the so-called ▶ curse of dimensionality. In a factored state representation, the number of states increases exponentially with the number of factors. This explosion of states produces two problems: it can be difficult to store a function over the state space, and even if the function can be stored, so much data is required to learn the function that learning is impractical.

The standard response to the curse of dimensionality is to apply function approximation to any function of state. This directly attacks the representation size, and also allows information from one state to affect another "similar" state allowing generalisation and learning.

While the addition of function approximation can significantly speed up learning, it also causes difficulty with convergence. Some types of function approximation will stop TD from converging at all. The resulting algorithms can either oscillate forever or approach infinite values. Other forms of approximation cause TD to converge to a estimate of long term reward which is only weakly related to the true long term reward (Gordon 1995; Boyan and Moore 1995; Baird 1995).

Most styles of function approximation used in conjunction with TD learning are parameterized, and the output is differentiable with respect to those parameters. Formally we have $V : \Theta \to \mathcal{S} \to \mathbb{R}$, where Θ is the space of possible parameter vectors, so that $V_\theta(s)$ is the value of V at state s with parameter vector θ, and $\nabla V_\theta(s)$ is the gradient of V with respect to θ at s. The TD update then becomes:

$$\begin{aligned} \delta_t &= \alpha(r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)) \\ \theta &\leftarrow \theta + \delta_t \nabla V_\theta(s_t) \end{aligned}$$

We describe three styles of approximation: state abstraction, linear approximation, and smooth general approximators (e.g., neural networks).

State abstraction refers to grouping states together and thereafter using the groups, or *abstract states*, instead of individual states. This can significantly reduce the amount of storage required for the value function as only values for abstract states need to be stored. It also preserves convergence results. A slightly more advanced form of state abstraction is the tile coding or CMAC (Albus 1981). In this type of function approximation, the state representation is assumed to be factored, i.e., each state is represented by a vector of values rather than a single scalar value. The CMAC represents the value function as the sum of separate value functions; one for each dimension of the state. Those individual dimensions can each have their own state abstraction. Again, TD has been shown to converge when used with a CMAC value function representation.

In general, any form of function approximation that forms a contraction mapping will converge when used with TD (see the entry on ▶ Markov Decision Processes). Linear interpolation is a contraction mapping, and hence converges. Linear extrapolation is not a contraction mapping and care needs to be taken when using general linear functions with TD. It has been shown that general linear function approximation used with TD will converge, but only when complete trajectories are followed through the state space (Tsitsiklis 1997).

It is not uncommon to use various types of back-propagation neural nets with TD, e.g., Tesauro's TD-gammon. More recently, TD algorithms have been proposed that converge for arbitrary differentiable function approximators (Papavassiliou 1999; Maei et al. 2009). These use more complex update techniques than those shown above.

Related Differencing Systems

TD learning was originally developed for use in environments where accurate models were unavailable. It has a close relationship with the theory of Markov Decision Processes where an accurate model is assumed. Using the notation $V(s_t) \rightsquigarrow V(s_{t+1})$ for a TD-style update that moves the value at $V(s_t)$ closer to the value at $V(s_{t+1})$ (including any discounting and intermediate rewards), we can now consider many possible updates.

As noted above, one way of applying TD to control is to use forward search. Forward search can be implemented using dynamic programming, and the result is closely related to TD. Let state $c(s)$ be the best child of state s in the forward search. We can then consider an update, $V(s) \rightsquigarrow V(c(s))$. If we let $l(s)$ be the best leaf in the forward search, we could then consider an update $V(s) \rightsquigarrow V(l(s))$. Neither of these updates consider the world after an actual state transition, only simulated state transitions, and so neither is technically a TD update.

Some work has combined both simulated time steps and real time steps. The TD-Leaf learning algorithm for alternative move games uses the $V(l(s_t)) \rightsquigarrow V(l(s_{t+1}))$ update rule (Baxter et al. 1998).

An important issue to consider when using forward search is whether the state distribution where learning takes place is different to the state distribution where the value function is used. In particular, if updates only occur for states the agent chooses to visit, but the search is using estimates for states that the agent is not visiting, then TD may give poor results. To combat this, the TreeStrap($\alpha - \beta$) algorithm for alternating move games updates all nodes in the forward search tree to be closer to the bound information provided by their children (Veness et al. 2009).

Biological Links

There are strong relationships between TD learning and the Rescorla–Wagner model of Pavlovian conditioning. The Rescorla–Wagner model is one way to formalize the idea that learning occurs when the co-occurence of two events is surprising rather than every time a co-occurence is experienced. The Δ_t value calculated in the TD update can be viewed as a measure of surprise. These findings appear to have a neural substrate in that dopamine cells react to reward when it is unexpected and to the predictor when the reward is expected (Schultz et al. 1997; Sutton 1990).

Cross-References

- ▸ Curse of Dimensionality
- ▸ Markov Decision Processes
- ▸ Markov Chain Monte Carlo
- ▸ Reinforcement Learning

Recommended Reading

Albus JS (1981) Brains, behavior, and robotics. BYTE, Peterborough. ISBN:0070009759

Auer P, Ortner R (2007) Logarithmic online regret bounds for undiscounted reinforcement learning. Neural and information processing systems (NIPS), Vancouver

Baird LC (1995) Residual algorithms: reinforcement learning with function approximation. In: Prieditis A, Russell S (eds) Machine learning: proceedings of the twelfth international conference (ICML95). Morgan Kaufmann, San Mateo, pp 30–37

Baxter J, Tridgell A, Weaver L (1998) KnightCap: a chess program that learns by combining TD(lambda) with game-tree search. In: Shavlik JW (ed.) Proceedings of the fifteenth international conference on machine learning (ICML'98). Morgan Kaufmann, San Francisco, pp 28–36

Bellman RE (1957) Dynamic programming. Princeton University Press, Princeton

Bertsekas DP, Tsitsiklis J (1996) Neuro-dynamic programming. Athena Scientific, Belmont

Boyan JA, Moore AW (1995) Generalization in reinforcement learning: safely approximating the value function. In: Tesauro G, Touretzky DS, Leen TK (eds) Advances in neural information processing systems, vol 7. MIT, Cambridge

Di Castro D, Meir R (2010) A convergent online single time scale actor critic algorithm. J Mach

Learn Res 11:367–410. http://jmlr.csail.mit.edu/papers/v11/dicastro10a.html
Gordon GF (1995) Stable function approximation in dynamic programming. Technical report CMU-CS-95-103. School of Computer Science, Carnegie Mellon University
Lagoudakis MG, Parr R (2003) Least-squares policy iteration. J Mach Learn Res 4:1107–1149. http://www.cs.duke.edu/~parr/jmlr03.pdf
Maei HR et al (2009) Convergent temporal-difference learning with arbitrary smooth function approximation. Neural and information processing systems (NIPS), pp 1204–1212. http://books.nips.cc/papers/files/nips22/NIPS2009_1121.pdf
Mahadevan S (1996) Average reward reinforcement learning: foundations, algorithms, and empirical results. Mach Learn 22:159–195. doi:10.1023/A:1018064306595
Papavassiliou VA, Russell S (1999) Convergence of reinforcement learning with general function approximators. International Joint Conference on Artificial Intelligence, Stockholm
Puterman ML (1994) Markov decision processes: discrete stochastic dynamic programming. Wiley series in probability and mathematical statistics. Applied probability and statistics section. Wiley, New York
Samuel AL (1959) Some studies in machine learning using the game of checkers. IBM J Res Dev 3(3):210–229
Schultz W, Dayan P, Read Montague P (1997) A neural substrate of prediction and reward. Science 275(5306):1593–1599. doi:10.1126/science.275.5306.1593
Sutton RS (1984) Temporal credit assignment in reinforcement learning. Ph.D. thesis, University of Massachusetts, Amherst
Sutton RS (1988) Learning to predict by the method of temporal differences. Mach Learn 3:9–44. doi:10.1007/BF00115009
Sutton RS, Barto AG (1990) Time-derivative models of Pavlovian reinforcement. In: Gabriel M, Moore J (eds) Learning and computational neuroscience: foundations of adaptive networks. MIT, Cambridge, pp 497–537
Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT, Cambridge
Sutton R, Tanner B (2004) Temporal difference networks. Neural and information processing systems (NIPS), Vancouver
Tesauro G (1995) Temporal difference learning and TD-gammon. Commun ACM 38(3):58–67
Tsitsiklis JN, Van Roy B (1997) An analysis of temporal-difference learning with function approximation. IEEE Trans Autom Control 42(5):674–690
Veness J et al (2009) Bootstrapping from game tree search. Neural and information processing systems (NIPS), Whistler
Watkins CJCH (1989) Learning with delayed rewards. Ph.D. thesis, Psychology Department, Cambridge University, Cambridge

Test Data

Synonyms

Evaluation data; Test instances

Definition

Test data are data to which a model is applied for the purposes of ▶ evaluation. When ▶ holdout evaluation is performed, test data are also called *out-of-sample data*, *holdout data*, or the *holdout set*.

Cross-References

▶ Test Set

Test Instances

▶ Test Data

Test Set

Synonyms

Evaluation data; Evaluation set; Test data

Definition

A test set is a ▶ data set containing data that are used for ▶ evaluation by a learning system. Where the ▶ training set and the test set contain disjoint sets of data, the test set is known as a ▶ holdout set.

Cross-References

▶ Data Set

Test Time

A learning algorithm is typically applied at two distinct times. Test time refers to the time when an algorithm is applying a learned model to make predictions. ▶ Training time refers to the time when an algorithm is learning a model from ▶ training data. ▶ Lazy learning usually blurs the distinction between these two times, deferring most learning until test time.

Test-Based Coevolution

Synonyms

Competitive coevolution

Definition

A coevolutionary system constructed to simultaneously develop solutions to a problem and challenging tests for candidate solutions. Here, individuals represent complete solutions or their tests. Though not precisely the same as *competitive coevolution*, there is a significant overlap.

Text Learning

▶ Text Mining

Text Mining

Dunja Mladenić
Artificial Intelligence Laboratory, Jožef Stefan Insitute, Ljubljana, Slovenia

Abstract

Text Mining also referred to as Data Mining on Text, has emerged at the intersection of several research areas, some focused on data analytics and others focused more on handling text data. This entry provides a definition of Text Mining and links it to related research areas, most of them included in this book.

Synonyms

Analysis of text; Data mining on text; Text learning

Definition

The term *text mining* is used analogous to ▶ Data Mining when the data is text. As there are some data specificities when handling text compared to handling data from ▶ databases, text mining has a number of specific methods and approaches. Some of these are extensions of data mining and machine learning methods, while others are rather text specific. Text mining approaches combine methods from several related fields, including machine learning, data mining, ▶ Information Retrieval, natural language processing, ▶ Statistical Learning, and ▶ Semantic Web. Basic text mining approaches are also extended to enable handling of different natural languages (▶ Cross-Lingual Text Mining) and are combined with methods for handling different data types, such as links and graphs (▶ Link Mining and Link Discovery, Graph Mining), images and video (multimedia mining), and sensor data. Adopting ▶ Stream Mining methods for text data enables analysis of text streams, such as news feed or social media texts. Text stream mining can be also combined with other types of data streams, such as sensor readings, economic indicators, and video, where time stamp and location of the data can play a crucial role in analytics.

Cross-References

- ▶ Cross-Lingual Text Mining
- ▶ Feature Construction in Text Mining
- ▶ Feature Selection in Text Mining
- ▶ Semi-Supervised Text Processing
- ▶ Stream Mining

- ▶ Text Mining for Advertising
- ▶ Text Mining for News and Blogs Analysis
- ▶ Text Mining for Spam Filtering
- ▶ Text Mining for the Semantic Web
- ▶ Text Visualization

Text Mining for Advertising

Massimiliano Ciaramita
Yahoo! Research Barcelona, Barcelona, Spain

Synonyms

Content match; Contextual advertising; Sponsored search; Web advertising

Definition

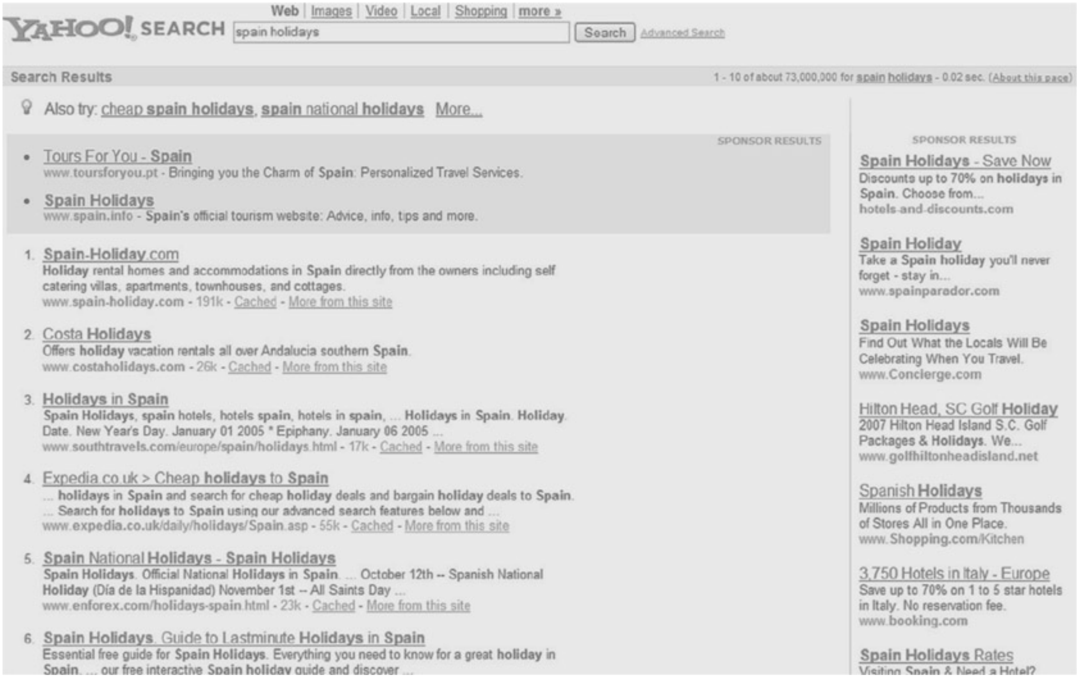
Text mining for advertising is an area of investigation and application of text mining and machine learning methods to problems such as Web advertising; e.g., automatically selecting the most appropriate ads with respect to a Web page, or query submitted to a search engine. Formally, the task can be framed as a ranking or matching problem where the unit of retrieval, rather than a Web page, is an advertisement. Most of the time ads have simple and homogeneous predefined textual structures, however, formats can vary and include audio and visual information. Advertising is a challenging problem due to several factors such as the economic nature of the transactions involved, engineering issues concerning scalability, and the inherent complexity of modeling the linguistic and multimedia content of advertisements.

Motivation and Background

The role of advertising in supporting and shaping the development of the Web has substantially increased over the past years. According to the Interactive Advertising Bureau (IAB), Internet advertising revenues in the USA totaled almost $8 billion in the first 6 months of 2006, a 36.7 % increase over the same period in 2005, the last in a series of consecutive growths. Search, i.e., ads placed by Internet companies in Web pages or in response to specific queries, is the largest source of revenue, accounting for 40 % of total revenue (Internet Advertising Bureau 2006). The most important categories of Web advertising are *keyword match*, also known as *sponsored search* or *paid listing*, which places ads in the search results for specific queries (see Fain and Pedersen 2006 for a brief history of sponsored search), and *content match*, also called *content-targeted advertising* or *contextual advertising*, which places ads in Web pages based on the page content. Figure 1 shows an example of sponsored search and ads are listed on the right side of the page.

Currently, most of the focus in Web advertising involves sponsored search, because matching based on keywords is a well-understood problem. Content match has greater potential for content providers, publishers, and advertisers, because users spend most of their time on the Web on content pages as opposed to search engine result pages. However, content match is a harder problem than sponsored search. Matching ads with query terms is to a certain degree straightforward, because advertisers themselves choose the keywords that characterize their ads that are matched against keywords chosen by users while searching. In contextual advertising, matching is determined automatically by the page content, which complicates the task considerably.

Advertising touches challenging problems concerning how ads should be analyzed, and how the accurately and efficiently systems select the best ads. This area of research is developing rapidly in information retrieval. How best to model the structure and components of ads, and the interaction between the ads and the contexts in that they appear are open problems. Information retrieval systems are designed to capture relevance, which is a basic concept in advertising as well. Elements of an ad such as text and images tend to be mutually relevant, and often (in print media for example) ads are placed in contexts that match a certain product at



1 - 10 of about 73,000,000 for spain holidays - 0.02 sec. (About this page)

Also try: cheap spain holidays, spain national holidays More...

- Tours For You - Spain
  www.toursforyou.pt - Bringing you the Charm of Spain: Personalized Travel Services.
- Spain Holidays
  www.spain.info - Spain's official tourism website: Advice, info, tips and more.

1. Spain-Holiday.com
   Holiday rental homes and accommodations in Spain directly from the owners including self catering villas, apartments, townhouses, and cottages.
   www.spain-holiday.com - 191k - Cached - More from this site
2. Costa Holidays
   Offers holiday vacation rentals all over Andalucia southern Spain.
   www.costaholidays.com - 26k - Cached - More from this site
3. Holidays in Spain
   Spain Holidays, spain hotels, hotels spain, hotels in spain, ... Holidays in Spain. Holiday. Date. New Year's Day. January 01 2005 * Epiphany. January 06 2005 ...
   www.southtravels.com/europe/spain/holidays.html - 17k - Cached - More from this site
4. Expedia.co.uk > Cheap holidays to Spain
   ... holidays in Spain and search for cheap holiday deals and bargain holiday deals to Spain. ... Search for holidays to Spain using our advanced search features below and ...
   www.expedia.co.uk/daily/holidays/Spain.asp - 55k - Cached - More from this site
5. Spain National Holidays - Spain Holidays
   Spain Holidays. Official National Holidays in Spain. ... October 12th -- Spanish National Holiday (Día de la Hispanidad) November 1st -- All Saints Day ...
   www.enforex.com/holidays-spain.html - 23k - Cached - More from this site
6. Spain Holidays. Guide to Lastminute Holidays in Spain
   Essential free guide for Spain Holidays. Everything you need to know for a great holiday in Spain. ... our free interactive Spain holiday guide and discover ...

Spain Holidays - Save Now
Discounts up to 70% on holidays in Spain. Choose from...
hotels-and-discounts.com

Spain Holiday
Take a Spain holiday you'll never forget - stay in...
www.spainparador.com

Spain Holidays
Find Out What the Locals Will Be Celebrating When You Travel.
www.Concierge.com

Hilton Head, SC Golf Holiday
2007 Hilton Head Island S.C. Golf Packages & Holidays. We...
www.golfhiltonheadisland.net

Spanish Holidays
Millions of Products from Thousands of Stores All in One Place.
www.Shopping.com/Kitchen

3,750 Hotels in Italy - Europe
Save up to 70% on 1 to 5 star hotels in Italy. No reservation fee.
www.booking.com

Spain Holidays Rates
Visiting Spain & Need a Hotel?

Text Mining for Advertising, Fig. 1 Ads ranked next to a search results page for the query "Spain holidays"

a topical level; e.g., an ad for sneakers placed on a sport news page. However, topical relevance is only one the basic parameters which determine a successful advertisement placement. For example, an ad for sneakers might be appropriate and effective on a page comparing MP3 players, because they may share a target audience (e.g., joggers) although they arguably refer to different topics, and it is possible they share no common vocabulary. Conversely, there may be ads that are topically similar to a Web page, but cannot be placed there because they are inappropriate. An example might be placing ads for a product in the page of a competitor.

The language of advertising is rich and sophisticated and can rely considerably on complex inferential processes. A picture of a sunset in an ad for life insurance carries a different implication than a picture of a sunset in an ad for beer. Layout and visual content are designed to elicit inferences, possibly hinging on cultural elements; e.g., the age, appearance, and gender of people in an ad affect its meaning. Adequate automatic modeling will likely involve, to a substantial degree, understanding the language of advertisement and the inferential processes involved (Vestergaard and Schroeder 1985). Today this seems beyond what traditional information retrieval systems are designed to cope with. In addition, the global context can be captured only partially by modeling the text alone. As the Web evolves into an immense infrastructure for social interaction and multimedia information sharing the need for modeling structured "content" becomes more and more crucial. This applies to information retrieval and specifically to advertising. For this reason, the problem of content match is of particular interest and opens new problems and opportunities for interdisciplinary research.

Today, contextual advertising, the most interesting sub-task from a mining perspective, consists mostly in selecting ads from a pool to match the textual content of a particular Web page. Ads provide a limited amount of text: typically a few keywords, a title, and brief description. The ad-placing system needs to identify relevant ads, from huge ad inventories, quickly and efficiently based on this very limited amount of information.

Current approaches have focused on augmenting the representation of the page to increase the chance of a match (Ribeiro-Neto et al. 2005), or by using machine learning to find complex ranking functions (Lacerda et al. 2006), or by reducing the problem of content match to that of sponsored search by extracting keywords from the Web page (Yih et al. 2006). All these approaches are based on methods that quantify the similarity between the ad and the target page on the basis of traditional information retrieval notions such as cosine similarity and *tf.idf* features. The relevance of an ad for a page depends on the number of overlapping words, weighted individually and independently as a function of their individual distributional properties in the collection of documents or ads.

Structure of Learning Problem

The typical elements of an advertisement are a set of *keywords*, a *title*, a *textual description* and a hyperlink pointing to page, the *landing page*, relative to a product or service, etc. In addition, an ad has an *advertiser id* and can be part of a *campaign*, i.e., a subset of all the ads with same advertiser id. This latter information can be used, for example, to impose constraints on the number of ads to display relative to the campaign or advertiser. While this is possibly the most common layout, it is important to realize that ads structure can vary significantly and include relevant audio and visual content.

In general, the learning problem for an ad-placing system can be formalized as a ranking task. Let $\mathcal{A}$ be a set of ads, $\mathcal{P}$ the set of possible pages, and $\mathcal{Q}$ the set of possible queries. In keyword match, the goal is to find a function $F : \mathcal{A} \times \mathcal{Q}$; e.g., a function that counts the number of individual common terms or n-grams of such terms. In content match, the objective is to find a function $F : \mathcal{A} \times \mathcal{P} \rightarrow \mathbb{R}$. The keyword match problem is to a certain extent straightforward and amounts to matching small set of terms defined manually by both the user and the advertiser. The content match task shares with the former task the peculiarities of the units of retrieval (the ads), but introduces new and interesting issues for text mining and learning because of the more complex conditioning environment, the Web page content, which needs to modeled automatically.

In general terms, an ad can be represented as a feature vector $\mathbf{x} = \Phi(a_i, p_j)$ such that $a_i \in \mathcal{A}$, $p_j \in \mathcal{P}$, and given a d-dimensional feature space $\mathcal{X} \subset \mathbb{R}^d$, $\Phi : \mathcal{A} \times \mathcal{P} \rightarrow \mathcal{X}$. In the traditional machine learning setting, one introduces a weight vector $\alpha \in \mathbb{R}^d$ which quantifies each feature's contribution individually. The vector's weights can be learned from manually edited rankings (Lacerda et al. 2006; Ribeiro-Neto et al. 2005) or from click-through data as in search results optimization (Joachims 2002). In the case of a linear classifier the score of an ad-target page pair x_i would be:

$$F(\mathbf{x}; \alpha) = \sum_{s=1}^{d} \alpha_s x_s. \tag{1}$$

Several methods can be used to learn similar or related models such as perceptron, SVM, boosting, etc. Constraints on the number of advertisers or campaigns could be easily implemented as post-ranking filters on the top of the ranked list of ads or included in a suitable objective function.

A basic model for ranking ads can be defined in the vector space model for information retrieval, using a ranking function based on cosine similarity, where ads and target pages are represented as vectors of terms weighted by fixed schemes such as *tf.idf*. If only one feature is used, the cosine based on *tf.idf* between the ad and the page, a standard vector space model baseline is obtained, which is at the base of the ad-placing ranking functions variants proposed by Ribeiro-Neto et al. (2005). Recent work has shown that machine learning-based models are considerably more accurate than such baselines. However, as in document retrieval, simple feature maps which include mostly coarse-grained statistical properties of the ad-page pairs, such as *tfidf-based* cosine, are the most desirable for efficiency and bias reasons. Properties of the different components of the ad can be used and weighted in different ways, and soft or hard constraints introduced to model the presence of the ads keyword in the Web page. The design space for ad-place systems is vast and still little explored. All systems presented so far in the literature make use of manually annotated data for training and/or evaluating a model.

Structure of Learning Systems

Web advertising presents peculiar engineering and modeling challenges and has motivated research in different areas. Systems need to be able to deal in real time with huge volumes of data and transactions involving billions of ads, pages, and queries. Hence several engineering constraints need to be taken into account; efficiency and computational costs are crucial factors in the choice of matching algorithms (The Yahoo! Research Team 2006). Ad-placing systems might require new global architecture design; e.g., Attardi et al. (2004) proposed an architecture for information retrieval systems that need to handle large-scale targeted advertising based on an information filtering model. The ads that appear on Web pages or search results pages will ultimately be determined taking into account the expected revenues and the price of the ads. Modeling the microeconomics factors of such processes is a complex area of investigation in itself (Feng et al. 2005).

Another crucial issue is the evaluation of the effectiveness of the ad-placing systems. Studies have emphasized the impact of the quality of the matching on the success of the ad in terms of click-through rates (Gallagher et al. 2001; Sherman and Deighton 2001). Although click-through rates (CTRs) provide a traditional measure of effectiveness, it has been found that ads can be effective even when they do not solicit any conscious response and that the effectiveness of the ad is mainly determined by the level of congruency between the ad and the context in which it appears (Yoo 2006).

Keyword Extraction Approaches

Since the query-based ranking problem is better understood than contextual advertising, one way of approaching the latter would be to represent the content page as a set of keywords and then ranking the ads based on the keywords extracted from the content page. Carrasco et al. (2003) proposed clustering of bi-partite advertiser-keyword graphs for keyword suggestion and identifying groups of advertisers. Yih et al. (2006) proposed a system for keyword extraction from content pages. The goal is to determine which keywords, or key phrases, are more relevant in a Web page. Yih et al. develop a supervised approach to this task from a corpus of pages where keywords have been manually identified. They show that a model learned with ▸ logistic regression outperforms traditional vector models based on fixed *tf.idf* weights. The most useful features to identify good keywords efficiently are, in this case, term frequency and document frequency of the candidate keywords, and particularly the frequency of the candidate keyword in a search engine query log. Other useful features include the similarity of the candidate with the page's URL and the length, in number of words, of the candidate keyword. In terms of feature representation thus, they propose a feature map $\Phi : \mathcal{A} \rightarrow \mathcal{Q}$, which represent a Web page as a set of keywords. The accuracy of the best learned system is 30.06 %, in terms of the top predicted keyword being in the set of manually generated keywords for a page, against 13.01 % of the simpler *tf.idf* based model. While this approach is simple to apply, it remains to be seen how accurate it is at identifying good ads for a page. It identifies potentially useful sources of information in automatically-generated keywords. An interesting related finding concerning keywords is that longer keywords, about four words long, lead to increased click-through rates (OneUpWeb 2005).

The Vocabulary Impedance Problem

Ribeiro-Neto et al. (2005) introduced an approach to content match which focuses on the vocabulary mismatch problem. They notice that there tends to be not enough overlap in the text of the ad and the target page to guarantee good accuracy; they call this the *vocabulary impedance problem*. To overcome this limitation they propose to generate an augmented representation of the target page by means of a Bayesian model previously applied to document retrieval (Ribeiro-Neto and Muntz 1996). The expanded vector representation of the target page includes a significant number of additional words which can potentially match some of the terms in the ad. They find that such a model improves over a standard vector space model baseline, evaluated by means of 11-point average precision on a test bed of 100 Web pages, from 0.168 to 0.253. One possible shortcoming of such an approach is that generating the augmented representation involves crawling a significant number of additional related pages. It has also been argued (Yih et al. 2006) that this model complicates pricing of the ads because the keywords chosen by the advertisers might not be present in the content of the matching page.

Learning with Genetic Programming

Lacerda et al. (2006) proposed to use machine learning to find good ranking functions for contextual advertising. They use the same data-set described in Ribeiro-Neto et al. (2005), but use part of the data for training a model and part for evaluation purposes. They use a genetic programming algorithm to select a ranking function which maximizes the average precision on the training data. The resulting ranking function is a nonlinear combination of simple components based on the frequency of ad terms in the target page, document frequencies, document length, and size of the collections. Thus, in terms of the feature representation defined earlier, they choose a feature map which extracts traditional features from the ad-page pair, but then apply then genetic programming methods to select complex nonlinear combinations of such features that maximize a fitness function based on average precision. Lacerda et al. (2006) find that the ranking functions selected in this way are considerably more accurate than the baseline proposed in Ribeiro-Neto et al. (2005); in particular, the best function selected by genetic programming achieves an average precision at position three of 0.508, against 0.314 of the baseline.

Semantic Approaches to Contextual Advertising

The most common approaches to contextual advertising are based on matching terms between the ad and the content page. Broder et al. (2007) notice that this approach (which they call the "syntactic—" model), can be improved by adopting a matching model which additionally takes into account topical proximity; i.e., a "semantic" model. In their model the target page and the ad are classified with respect to a taxonomy of topics. The similarity of ad and target page estimated by means of the taxonomy provides an additional factor in the ads ranking function. The taxonomy, which has been manually built, contains approximately 6,000 nodes, where each node represents a set of queries. The concatenation of all queries at each node is used as a meta-document, ads and target pages are associated with a node in the taxonomy using a nearest neighbor classifier and *tf. idf* weighting. The ultimate score of an ad a_i for a page p is a weighted sum of the taxonomy similarity score and the similarity of a_i and p based on standard syntactic measures (vector cosine). On evaluation, Broder et al. (2007) report a 25 % improvement for mid-range recalls of the syntactic-semantic model over the pure syntactic one.

Cross-References

- ▶ Boosting
- ▶ Genetic Programming
- ▶ Information Retrieval
- ▶ Model Space
- ▶ Precision
- ▶ Support Vector Machines
- ▶ TF–IDF

Recommended Reading

Attardi G, Esuli A, Simi M (2004) Best bets, thousands of queries in search of a client. In: Proceedings of the 13th international conference on World Wide Web, alternate track papers & posters. ACM Press, New York

Broder A, Fontoura M, Josifovski V, Riedel L (2007) A semantic approach to contextual advertising. In: Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval. ACM Press, New York

Carrasco JJ, Fain D, Lang K, Zhukov L (2003) Clustering of bipartite advertiser-keyword graph. In: Workshop on clustering large datasets, IEEE conference on data mining. IEEE Computer Society Press, New York

Fain D, Pedersen J (2006) Sponsored search: a brief history. In: Proceedings of the 2nd workshop on sponsored search auctions, Ann Arbor. Web Publications

Feng J, Bhargava H, Pennock D (2005, forthcoming) Implementing sponsored search in web search engines: computational evaluation of alternative mechanisms. Inf J Comput

Gallagher K, Foster D, Parsons J (2001) The medium is not the message: advertising effectiveness and content evaluation in print and on the Web. J Advert Res 41(4):57–70

Internet Advertising Bureau (IAB) (2006) IAB internet advertising revenue report. http://www.iab.net/resources/adrevenue/pdf/IAB_PwC%202006Q2.pdf

Joachims T (2002) Optimizing search engines using clickthrough data. In: Proceedings of the ACM conference on knowledge discovery and data mining (KDD). ACM Press, New York

Lacerda A, Cristo M, Gonçalves MA, Fan W, Ziviani N, Ribeiro-Neto B (2006). Learning to advertise. In: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval. ACM Press, New York, pp 549–556

OneUpWeb (2005) How keyword length affects conversion rates. http://www.oneupweb.com/landing/keywordstudy_landing.htm

Ribeiro-Neto B, Cristo M, Golgher PB, de Moura ES (2005) Impedance coupling in content-targeted advertising. In: Proceedings of the 28th annual international ACM SIGIR conference on research and development in information retrieval. ACM Press, New York, pp 496–503

Ribeiro-Neto B, Muntz R (1996) A belief network model for IR. In: Proceedings of the 19th annual international ACM SIGIR conference on research and development in information retrieval. ACM Press, New York, pp 253–260

Sherman L, Deighton J (2001) Banner advertising: measuring effectiveness and optimizing placement. J Interact Mark 15(2):60–64

The Yahoo! Research Team (2006) Content, metadata, and behavioral information: directions for Yahoo! Research. IEEE Data Eng Bull 29(4):10–18

Vestergaard T, Schroeder T (1985) The language of advertising. Blackwell, Oxford

Yih W, Goodman J, Carvalho VR (2006) Finding advertising keywords on web pages. In: Proceedings of the 15th international conference on World Wide Web. ACM Press, New York, pp 213–222

Yoo CY (2006) Preattentive processing of web advertising. Ph.D. thesis, University of Texas, Austin

Text Mining for News and Blogs Analysis

Bettina Berendt
KU Leuven, Leuven, Belgium

Abstract

News and blogs are temporally indexed online texts and play a key role in today's information distribution and consumption. News communicate selected information on current events, written by professional or citizen journalists; blogs are updated publications on the Web that span a much wider range of topics, styles, and authors. Particularly important in recent years have been microblogs such as Twitter. The entry gives an overview of how text mining (for tasks such as description, classification, prediction, search, recommendation, or summarization) is applied to analyze the textual parts of news and blogs, extracting topics, events, opinions, sentiments, and other aspects of content. Often, textual analysis is complemented by the analysis of further data such as the social network of authors and readers. The properties of news and blogs data structures and language use require methods for preprocessing and analyzing that are tailored to news and (micro)blogs, and the tasks often profit from an interactive approach in which the user plays an active role in sensemaking. The methods are deployed in a wide range of applications and services.

Definition

News is "the communication of selected information on current events," where the selection is guided by "newsworthiness" or "what interests the public." News are also stories, from which the reader usually expects answers to the five Ws: who, what, when, where, and why, to which a "how" is often added. News-style writing – as opposed to, for example, commentary writing – generally strives for objectivity and/or neutrality (the representation of different views on the event).

In this content-centric sense, news can be written/authored and published by professional journalists and news outlets (such as newspapers or radio or TV stations) but also by anyone else and in any other form, often called *citizen journalism*: "an alternative and activist form of newsgathering and reporting that functions outside mainstream media institutions, often as a response to shortcomings in the professional journalistic field, that uses similar journalistic practices but is driven by different objectives and ideals and relies on alternative sources of legitimacy than traditional or mainstream journalism." (Radsch 2013, p. 159). However, news, or mainstream (media) news, is also often thought of in a source-centric way: reports authored by professional journalists in mainstream media institutions, as opposed to reporting from citizen journalists (or anyone else) who generally publish on the Web, in the form of blogs with a certain form of periodicity.

A *blog* is a (more or less) frequently updated publication on the Web, sorted in reverse chronological order of the constituent blog posts. Blog content may reflect any interest including journalistic, personal, corporate, and many others. Early blog posts (late 1990s) tended to be published on content management platforms without length restrictions; with the success of Twitter and similar *microblogging* platforms, much blogging (and of blog mining) has shifted to short posts (e.g., 140 characters on Twitter.com and Weibo.cn, although the latter's Chinese characters allow for much more complex messages). Twitter in particular has attained a major worldwide role in the fast diffusion of news (or short summaries and statements, enriched by hyperlinks to more text and other media), with citizen journalists, mainstream media themselves, politicians, and others being the publishers (Kwak et al. 2010). Current research in blog mining and the remainder of the present entry reflect this dominance of (a) news or news-related content and (b) microblog format. In addition, blog mining overlaps with *social-media mining* (Zafarani et al. 2014). In particular, the *social graph* of a microblogger allows the mining analyst to track the blogger's sources and readers/"followers" along with the contents.

News and blogs consist of textual and (in some cases) pictorial content and, when Web-based, may contain additional content in any other format (e.g., video, audio) and hyperlinks. They are indexed by time and structured into smaller units: news media into articles and blogs into blog posts. In most news and blogs, textual content dominates. Therefore, text analysis is the most often applied form of knowledge discovery. This comprises tasks and methods from data/text mining, ▶ information retrieval, and related fields. In accordance with the increasing convergence of these fields, this entry refers to all of them as ▶ *text mining*. The present entry will illustrate the overlap with/use of these fields and highlight the specifics that derive from the domain, including data, tasks, users, and use cases.

Motivation and Background

News and blogs are today's most common sources for learning about current events and also, in the case of blogs, for uttering opinions about current events. In addition, they may deal with topics of more long-term interest. Both reflect and form societies', groups', and individuals' views of the world, fast or even instantaneous with the events triggering the reporting. However, there are differences between these two types of media regarding authoring, content, and form. News is generally authored by people with journalistic training who abide by journalistic standards regarding the style and language of reporting. Topics and ways of reporting are circumscribed by general societal consensus and the policies of the news provider.

In contrast, everybody with Internet access can start a blog, and there are no restrictions on content and style (beyond the applicable types of censorship). Thus, blogs offer end users a wider range of topics and views on them.

These application characteristics lead to various linguistic and computational challenges for text mining analyses of news and blogs:

- Indexing, taxonomic categorization, partial redundancy, and data streams: News is indexed by time and by source (news agency or provider). In a multisource corpus, many articles published at about the same time (in the same or in other languages) describe the same events. Over time, a story may develop in the articles. Such multiple reporting and temporal structures are also observed for popular topics in blogs.
- Language and meaning: News is written in clear, correct, "objective," and somewhat schematized language. Usually, the start of a news article summarizes the whole article (feeds are a partial analogue of this in blogs). Information from external sources such as press agencies is generally reused rather than referenced. In sum, news makes fewer assumptions about the reader's background and context knowledge than many other texts.
- Nonstandard language and subjectivity: The language in blogs ranges from high-quality, "news-like" language via poor-quality, restricted-code language with many spelling and grammatical errors to creative, sometimes literary, language. A blog may employ high-quality language but operate outside the news genre or across journalistic genres (e.g., combining current-events reporting with commentary and background information). Jargon is very common in blogs, and new linguistic developments are adopted far more quickly than could be reflected in external resources such as lexica. Many blog authors strive not for objectivity but for subjectivity and emotionality.
- Thematic diversity and new forms of categorization: News are generally categorized by topic area ("politics," "business," etc.). In contrast, a blog author may choose to write about differing, arbitrary topics. When blogs are labeled, it is usually not with reference to a stable, taxonomic system, but with an arbitrary number of tags: free-form, often informal labels chosen by the user.
- Context and its impact on content and meaning: The content of a blog (post) is often not contained in the text alone. Rather, blog software supports "Web" and "Social Web" behavior, and bloggers practice it: multiway communication rather than broadcasting and semantics-inducing referencing of both content and people. Specifically, hyperlinks to other resources provide not only context but also content, as do links to and from cited resp. citing people/sources. The latter evolved from "blogrolls" resp. "trackback links" in early blog software to "followees" and "retweet" links resp. "followers" in platforms such as Twitter.

Structure of the Learning System

Tasks

From a text mining point of view, tasks can be grouped by different criteria:

- Basic task and type of result: description, classification, and prediction (supervised or unsupervised, includes, for example, topic identification, tracking, and/or novelty detection, spam detection), search (ad hoc or filtering), recommendation (of blogs, blog posts, or (hash-)tags), and summarization
- Higher-order characterization to be extracted: especially topic or event, opinion, or sentiment
- Time dimension: nontemporal, temporal (stream mining), and multiple streams (e.g., in different languages; see cross-lingual ▶ text mining)
- User adaptation: none (no explicit mention of user issues and/or general audience), customizable, and personalized

Real-world applications increasingly employ selections or, more often, combinations of these tasks by their intended users and use cases, in particular:

- News aggregators allow laypeople and professional users (e.g., journalists) to see “what’s in the news” and to compare different sources’ texts on one story. Reflecting the presumption that news (especially mainstream news – sources for news aggregators are usually whitelisted) are mostly objective/neutral, these aggregators focus on topics and events. News aggregators are now provided by all major search engines.
- Social-media monitoring tools allow laypeople and professional users to track not only topical mentions of a keyword or named entity (e.g., person, brand) but also aggregate sentiment toward it. The focus on sentiment reflects the perceptions that even when news-related, social-media content tends to be subjective and that studying the blogosphere is therefore an inexpensive way of doing market research or public opinion research. The whitelist here is usually the platforms (e.g., Twitter, Tumblr, LiveJournal, Facebook) rather than the sources themselves, reflecting the huge size and dynamic structure of the blogosphere/the Social Web. The landscape of commercial and free social-media monitoring tools is wide and changes frequently; up-to-date overviews and comparisons can easily be found on the Web.
- Emerging application types include text mining not of but for journalistic texts, in particular natural language generation in domains with highly schematized event structures and reporting, such as sports and finance reporting (e.g., Allen et al. 2010, narrativescience.com) and social-media monitoring tools for helping journalists find sources (Diakopoulos et al. 2012).

Some tools have dashboard-style interfaces and complex data graphics, which may be most interesting for some professional users. However, the increasing move especially of casual users toward mobile devices with small screens has led to most applications showing original content and mining output that consists of (especially short) texts and a small number of (especially numeric) analytics.

Solution Approaches

Standardization: Tasks, Datasets, and APIs

The development of methods for mining news, blogs, and social media in general has profited from *standard datasets* and *standard tasks* and *competitions*. Prominent examples are the Reuters-21578 dataset, which is not only a collection of newswire articles but also the most classical dataset for text mining in general (https://archive.ics.uci.edu/ml/datasets/Reuters-21578+Text+Categorization+Collection); the larger and also multilingual RCV1, RCV2, and TRC2 datasets (http://trec.nist.gov/data/reuters/reuters.html); the blog datasets provided by the International Conference on Weblogs and Social Media (ICWSM, http://www.icwsm.org); and the SNAP datasets (https://snap.stanford.edu/data). The Topic Detection and Tracking (TDT) research program and workshops (http://www.itl.nist.gov/iad/mig/tests/tdt; Allan 2002) were essential in the formation of news mining as a research topic. Important tasks and competitions that are ongoing, and that also offer important datasets, include the Text Retrieval Conference (TREC, http://trec.nist.gov) and the Text Analysis Conference (TAC, http://www.nist.gov/tac), formerly Document Understanding Conference (DUC, http://duc.nist.gov). The history of tracks/tasks over time in these conferences also illustrates how fields have matured or become less relevant; for example, “blog tracks” have been replaced since 2010 by “microblog tracks,” and “topic detection” has given way to “event detection.”

Standard datasets are one answer to a central problem in news, blogs, and social-media mining in general. Since most platforms are commercial, they restrict access to their current or archived editions. Other platforms offer a free API but make it return a sample whose representativeness and/or even sampling criteria are not known;

this can affect mining results severely (Morstatter et al. 2013). In addition, the terms of use present a challenge for creating reusable datasets (for a solution approach, see McCreadie et al. 2012).

A further caveat concerns all social-media mining results: In general, APIs only give access to "public" posts and not to posts that users have set to "private" or otherwise limited to a restricted audience. In addition, having gained access to an individual's online communication does not mean one may use or process it. Thus, privacy and data protection considerations limit the uses of social media for research, and they require careful interpretations of the results: these may be representative of the public utterances of users, but not all of their online communication.

The Modeling Phase of Text Mining

Solution approaches are based on general data mining methods and adapted to the conceptual specifics of news and blogs and their mining tasks (see list of tasks above). Methods include (document) ▶ classification and ▶ clustering and latent-variable techniques such as (P)LSA or LDA (cf. ▶ feature construction; specifically for an overview of topic models, see Blei 2012), ▶ mixture models, ▶ time series, and ▶ stream mining methods.

Named-entity recognition (e.g., Atkinson and Van der Goot 2009; Ritter et al. 2011; Li et al. 2012) is an important part or companion of tasks such as topic detection or text enrichment (e.g., Štajner et al. 2010). Topic tracking and event threading are used to follow a news story unfolding over time (e.g., Shahaf and Guestrin 2010), and especially for the purposes of summarization over time, special attention is paid to *bursty* topics or events (term introduced by Kleinberg 2002; see Subašić and Berendt 2013 for further references and empirical comparison), i.e., those that are marked by "spikes" in the frequency or other weight of reporting at certain points in time.

Information extraction can help to extract the *event(s)* of a news story. Events involve named entities (e.g., people and locations), a time, and a characterization of what the event is about. Information extraction can leverage background ontologies (e.g., Kuzey et al. 2014). This covers the first four of the "five Ws" of a news story; the "why" and "how" at present remain to be extracted by human readers from the original text (which is therefore generally accessible from platforms; see remarks on semiautomatic sensemaking below). Clustering can be useful for the extraction of events from multilingual sources (Leban et al. 2014). Regularities in how reporting (or the world?) evolves have also been used for predicting events from news (Radinsky and Horvitz 2013). The brevity of microblogs combined with the speed and volume of their streams poses special challenges for event detection (McCreadie et al. 2013).

Sentiment analysis and *opinion mining* are key especially for analyzing blogs and other social media (see overviews in Feldman 2013; Pang and Lee 2007; Potts 2013), and they are evolving toward more sophisticated methods that take syntactic structure and background knowledge/semantics into account (e.g., Gangemi et al. 2014). Sentiment analysis and opinion mining are designed to detect and classify "subjective" content and as such describes (some) social-media content well. It can also be appropriate for "subjective" journalistic genres such as commentary. However, this does not mean that news is really – or can ever be truly – objective. The often subtle and often subconscious structures, backgrounds, and convictions that express themselves in how a news story is told are referred to as media bias or framing, and text mining has begun to address them (e.g., Recasens et al. 2013; Pollak et al. 2011; Odijk et al. 2013).

Further classification tasks that are specifically relevant for news and blogs are generally solved with features that are characteristic of the domain and/or can be easily extracted from its data. They include (a) geolocation (e.g., Hale et al. 2012), (b) recommendation (e.g., tracking multiple topics over time in news, personalized to a user whose interests may change over time was developed by Pon et al. 2007; an approach for microblogs was proposed by Ren et al. 2013), and (c) spam detection and blocking (Kolari et al. 2006; for a general overview, see Castillo and Davison 2011).

Text summarization (for an overview, see Fiori 2014; specifically for microblogs, see Mackie et al. 2014) is a key technique for helping users to get an overview of (a) a single document's key messages or (b) a multitude by different documents, often from different sources that in turn may have copied from one another. Today, most summarizations are extractive, either extracting key sentences or non-sentence structures such as graphs. In real-world applications, even simpler forms are still predominant, including the extraction of single terms based, for example, on frequency and their display in tag clouds and the use of the first sentences of news articles that, by journalistic writing conventions, are designed to summarize the text. Abstractive summarization involves the generation of natural language, which remains a hard problem. Today, it is used mostly for text genres that are highly schematized, such that templates can be used and filled with the entities/constants relevant to the story at hand (see "Emerging application types" above).

Texts, or text summaries, can be represented not only as bags of words, sets of topics or events but also as graphs in which words and/or named entities stand in multiple relations to one another (see Berendt et al. 2014, for examples and further references). (Shallow) semantic parsing is often used to extract triples (e.g., subject-predicate-object statements) (e.g., Štajner et al. 2010; Sudhahar et al. 2015).

Text-based modeling can be enhanced by (e.g., social) *network* structure (e.g., Mei et al. 2008) (cf. ▶ link mining and link discovery). The analysis of how the actors in a network influence one another is important for the domain of news and social media (Guille et al. 2013). Such analyses are applied not only to individual text producers but more often to whole domains. One general question is how blogs and news, viewed in the aggregate, refer to and contextualize each other (e.g., Gamon et al. 2008; Berendt and Trümper 2009; Leskovec et al. 2009).

Specifics of Data Understanding, Data Cleaning, and Data Preparation

Data cleaning is similar to that of other online documents; in particular, it requires the provision or learning of wrappers for removing mark-up elements. Analysis methods that focus on text mining usually ignore hypermedia elements such as photographs and videos or use only their metadata.

While news texts employ standard language and can be handled with general-purpose text-analysis software, the *language* of (micro-)blogs requires specific lexica (e.g., containing the frequently used emoticons), abbreviation expansion and grammatical rules, and similar techniques (see "Noah's ARK" at http://www.ark.cs.cmu.edu/TweetNLP/ for a suite of tools and references), and linguists have found that rather than being "wrong" and ungrammatical, microblogs are evolving toward new systems that resemble spoken language and indicate nuances such as geographical region (Eisenstein 2015). Like other social media, they often contain irony and other indirect uses of language for expressing appreciation or discontent (e.g., Veale and Hao 2010), and this remains a major stumbling block for the machine understanding of these texts.

The *semi-structured* nature of blogs and news can give valuable cues for understanding. For example, the format elements "timestamp" and "number of comments" can be treated as indicators of increased topical relevance and likelihood of being opinionated, respectively (Mishne 2007). A combination of text clustering and *tag* analysis can serve to identify topics as well as the blogs that are on topic and likely to retain this focus over time (Hayes et al. 2007). Twitter *hashtags* have been used, for example, as indicators of sentiment (Wang et al. 2011).

Like other online texts, news and blogs make frequent use of *hyperlinks*, and the content of linked materials may be necessary even for a human reader to understand a post. This is particularly true for microblogs that are often mere pointers to a URL, or a URL plus a short comment. Many mining methods therefore enrich the text by, for example, the contents of referenced URLs (e.g., Abel et al. 2011). *Semantic enrichment* can also utilize external (semi-)structured data; for example, Wikification can add context information to microblogs by drawing on Wikipedia or DBPedia (e.g., Cheng and Roth 2013). All these methods can help to enrich and to disambiguate meaning.

The Importance of Interactive Tools for Semi-automatic Sensemaking

Like most of text mining, machine analyses of news, blogs, and other social media are a first step in a process of human sensemaking, whether for news consumers or for news producers. It is therefore imperative to provide them with interfaces that support further steps. Thus, tools for news consumers (such as news aggregators) typically provide links to the original articles. Tools for news producers show statistics (such as aggregate opinions of "the crowd" or properties of one potential source) as an information for journalists, and topics or events detected in corpora are generally a starting point for a story, but not a story in and of themselves. Reading, understanding, and writing news and blogs can probably never be totally automated. One reason for this is that different people read a given text differently, which is well known in social science media research but still often neglected in computational research – maybe because it requires us to question key methodological concepts of text mining such as "the ground truth." Interactive tools for story detection and tracking have been proposed as an answer to this dilemma (Berendt et al. 2014), and drag-and-drop story editors are used to create one's own new story (storify.com).

In addition, text mining as a method for dealing with large data volumes is often in competition with or combined with human intelligence for doing the same. Thus, for example, the contributions from many (often unpaid) volunteers and interface elements such as voting constitute the "social news aggregator" reddit.com, and Twitter's "retweeting" is a major, and human-led, way in which tweets are fed into, and develop influence across, multiple sub-networks formed by the platform's users. In these human-machine collaborations, the algorithms employed by a platform however are not neutral companions, but shape how users perceive others' opinions, which in turn affects their further posting behavior. For example, Twitter's "trending topics" algorithm rewards bursty topics (cf. Wilson 2013). This implies that even a topic contained in many tweets can, if the interest over time remains stable, disappear from the trending topics and thereby from public visibility. The implications of such algorithmic decisions on user choices and perceptions as well as public decisions and policy are a new research topic that will be relevant not only for text mining.

Recommended Reading

Abel F, Gao Q, Houben G-J, Tao K (2011) Semantic enrichment of Twitter posts for user profile construction on the social web. In: Proceedings of ESWC (2), pp 375–389

Allan J (ed) (2002) Topic detection and tracking: event-based information organization. Kluwer Academic Publishers, Norwell

Allen ND, Templon JR, McNally PS, Birnbaum L, Hammond K (2010) StatsMonkey: a data-driven sports narrative writer. In: Proceedings of 2010 AAAI fall symposium series. AAAI Press. http://www.aaai.org/ocs/index.php/FSS/FSS10/paper/view/2305

Atkinson M, Van der Goot E (2009) Near real time information mining in multilingual news. In: Proceedings of the 18th international conference on World Wide Web (WWW'09). ACM, New York, pp 1153–1154

Berendt B, Last M, Subašić I, Verbeke M (2014) New formats and interfaces for multi-document news summarization and its evaluation. In: Fiori, pp 231–255

Berendt B, Trümper D (2009) Semantics-based analysis and navigation of heterogeneous text corpora: the Porpoise news and blogs engine. In: Ting I-H, Wu H-J (eds) Web mining applications in e-commerce and e-services. Springer, Berlin

Blei DM (2012) Probabilistic topic models. Commun ACM 55(4):77–84

Castillo C, Davison BD (2011) Adversarial web search. Found Trends Inf Retr 4(5):377–486. doi:10.1561/1500000021

Cheng X, Roth D (2013) Relational inference for Wikification. In: Proceedings of EMNLP 2013, pp 1787–1796

Diakopoulos N, De Choudhury M, Naaman M (2012) Finding and assessing social media information sources in the context of journalism. In: Proceedings of CHI 2012. ACM, pp 2451–2460

Eisenstein J (2017) Written dialect variation in online social media. In: Boberg C, Nerbonne J, Watt D (eds) The handbook of dialectology. Wiley-Blackwell, Hoboken. Preprint available at http://www.cc.gatech.edu/jeisenst/papers/dialectology-chapter.pdf

Feldman R (2013) Techniques and applications for sentiment analysis. Commun ACM 56(4):82–89

Fiori A (ed) (2014) Innovative document summarization techniques: revolutionizing knowledge understanding. IGI Global, Hershey

Gamon M, Basu S, Belenko D, Fisher D, Hurst M, König AC (2008) BLEWS: using blogs to provide context for news articles. In: Adar E, Hurst M, Finin T, Glance N, Nicolov N, Tseng B, Salvetti F (eds) Proceedings of the second international conference on weblogs and social media (ICWSM'08), Seattle/Menlo Park. http://www.aaai.org/Papers/ICWSM/2008/ICWSM08-015.pdf

Gangemi A, Presutti V, Reforgiato Recupero D (2014) Frame-based detection of opinion holders and topics: a model and a tool. IEEE Comput Intell Mag 9(1):20–30

Guille A, Hacid H, Favre C, Zighed DA (2013) Information diffusion in online social networks: a survey. SIGMOD Rec 42(2):17–28

Hale S, Gaffney D, Graham M (2012) Where in the world are you? Geolocation and language identification in Twitter. In: Proceedings of ICWSM'12, pp 518–521

Hayes C, Avesani P, Bojars U (2007) An analysis of bloggers, topics and tags for a blog recommender system. In: Berendt B, Hotho A, Mladeni D, Semeraro G (eds) From web to social web: discovering and deploying user and content profiles. LNAI 4737. Springer, Berlin

Kleinberg JM (2002) Bursty and hierarchical structure in streams. In: Proceedings of SIGKDD 2002, pp 91–101

Kolari P, Java A, Finin T, Oates T, Joshi A (2006) Detecting spam blogs: a machine learning approach. In: Proceedings of the 21st national conference on artificial intelligence. AAAI, Boston

Kuzey E, Vreeken J, Weikum G (2014) A fresh look on knowledge bases: distilling named events from news. In: Proceedings of CIKM 2014, pp 1689–1698

Kwak H, Lee C, Park H, Moon S (2010) What is Twitter, a social network or a news media? In: Proceedings of WWW. ACM, pp 591–600

Leban G, Fortuna B, Brank J, Grobelnik M (2014) Event registry: learning about world events from news. In: Proceedings of WWW 2014 (companion volume), pp 107–110

Leskovec J, Backstrom L, Kleinberg J (2009) Meme-tracking and the dynamics of the news cycle. In: Elder IV JF, Fogelman-Soulié F, Flach PA, Zaki MJ (eds) Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining, Paris/New York

Li C, Weng J, He Q, Yao Y, Datta A, Sun A, Lee B-S (2012) TwiNER: named entity ecognition in targeted Twitter stream. In: Proceedings of the 35th international ACM SIGIR conference on research and development in information retrieval (SIGIR'12). ACM, New York, pp 721–730. doi:10.1145/2348283.2348380

Mackie S, McCreadie R, Macdonald C, Ounis I (2014) Comparing algorithms for microblog summarisation. In: Proceedings of CLEF 2014, pp 153–159

McCreadie R, Macdonald C, Ounis I, Osborne M, Petrovic S (2013) Scalable distributed event detection for Twitter. In: Proceedings of BigData conference 2013, pp 543–549

McCreadie R, Soboroff I, Lin J, Macdonald C, Ounis I, McCullough D (2012) On building a reusable Twitter corpus. In: Proceedings of the 35th international ACM SIGIR conference on research and development in information retrieval (SIGIR'12). ACM, New York, pp 1113–1114. doi:10.1145/2348283.2348495

Mei Q, Cai D, Zhang D, Zhai C (2008) Topic modeling with network regularization. In: Huai J, Chen R (eds) Proceeding of the 17th international conference on world wide web (WWW'08), Beijing/New York. doi:10.1007/978-0-387-30164-8_827

Mishne G (2007) Using blog properties to improve retrieval. In: Glance N, Nicolov N, Adar E, Hurst M, Liberman M, Salvetto F (eds) Proceedings of the international conference on weblogs and social media (ICWSM), Boulder. http://www.icwsm.org/papers/paper25.html

Morstatter F, Pfeffer J, Liu H, Carley KM (2013) Is the sample good enough? Comparing data from Twitter's streaming API with Twitter's firehose. In: Proceedings of ICWSM 2013. http://www.aaai.org/ocs/index.php/ICWSM/ICWSM13/paper/view/6071

Odijk D, Burscher B, Vliegenthart R, de Rijke M (2013) Automatic thematic content analysis: finding frames in news. In: Social informatics 2013. LNCS 8238. Springer, Berlin, pp 333–345

Pang B, Lee L (2007) Opinion mining and sentiment analysis. Found Trends Inf Retr 2(1–2):1–135

Pollak S, Coesemans R, Daelemans W, Lavraè N (2011) Detecting contrast patterns in newspaper articles by combining discourse analysis and text mining. Pragmatics 21(4):647–683

Pon RK, Cardenas AF, Buttler D, Critchlow T (2007) Tracking multiple topics for finding interesting articles. In: Berkhin P, Caruana R, Wu X (eds)

Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining, San Jose/New York

Potts (2013) Introduction to sentiment analysis. (slide set). http://www.stanford.edu/class/cs224u/slides/2013/cs224u-slides-02-26.pdf. Retrieved 15 Feb 2015

Radinsky K, Horvitz E (2013) Mining the web to predict future events. In: Proceedings of WSDM 2013, pp 255–264

Radsch CC (2013) Digital dissidence & political change: cyberactivism and citizen journalism in Egypt. Doctoral Dissertation, School of International Service, American University. Available at SSRN:http://ssrn.com/abstract=2379913

Recasens M, Danescu-Niculescu-Mizil C, Jurafsky D (2013) Linguistic models for analyzing and detecting biased language. In: Proceedings of ACL

Ren Z, Liang S, Meij E, de Rijke M (2013) Personalized time-aware tweets summarization. In: Proceedings of the 36th international ACM SIGIR conference on research and development in information retrieval (SIGIR'13). ACM, New York, pp 513–522. doi:10.1145/2484028.2484052

Ritter A, Clark S, Mausam, Etzioni O (2011) Named entity recognition in tweets: an experimental study. In: Proceedings of the conference on empirical methods in natural language processing (EMNLP'11). Association for Computational Linguistics, Stroudsburg, pp 1524–1534

Shahaf D, Guestrin C (2010) Connecting the dots between news articles. In: Proceedings of SIGKDD 2010, pp 623–632

Sudhahar S, de Fazio G, Franzosi R, Cristianini N (2015) Network analysis of narrative content in large corpora. Nat Lang Eng 21(1): 81–112

Štajner T, Rusu D, Dali L, Fortuna B, Mladenic D, Grobelnik M (2010) A service oriented framework for natural language text enrichment. Informatica (Ljublj.) 34(3):307–313

Subašić I, Berendt B (2013) Story graphs: tracking document set evolution using dynamic graphs. Intell Data Anal 17(1):125–147

Veale T, Hao Y (2010) Detecting ironic intent in creative comparisons. In: Coelho H, Studer R, Wooldridge M (eds) Proceedings of the 2010 conference on ECAI 2010: 19th European conference on artificial intelligence. IOS Press, Amsterdam, pp 765–770

Wang X, Wei F, Liu X, Zhou M, Zhang M (2011) Topic sentiment analysis in Twitter: a graph-based hashtag sentiment classification approach. In: Berendt B, de Vries A, Fan W, Macdonald C, Ounis I, Ruthven I (eds) Proceedings of the 20th ACM international conference on information and knowledge management (CIKM'11). ACM, New York, pp 1031–1040. doi:10.1145/2063576.2063726

Wilson R (2013) Trending on Twitter: a look at algorithms behind trending topics. Ignite social media blog. http://www.ignitesocialmedia.com/twitter-marketing/trending-on-twitter-a-look-at-algorithms-behind-trending-topics/. Retrieved 15 Feb 2015

Zafarani R, Abbasi MA, Liu H (2014) Social media mining: an introduction. Cambridge University Press, Cambridge

Text Mining for Spam Filtering

Aleksander KoŁcz
Microsoft One Microsoft Way, Redmond, WA, USA

Synonyms

Commercial Email Filtering; Junk email filtering; Spam detection; Unsolicited commercial email filtering

Definition

Spam filtering is the process of detecting unsolicited commercial email (UCE) messages on behalf of an individual recipient or a group of recipients. Machine learning applied to this problem is used to create discriminating models based on labeled and unlabeled examples of spam and nonspam. Such models can serve populations of users (e.g., departments, corporations, ISP customers) or they can be personalized to reflect the judgments of an individual. An important aspect of spam detection is the way in which textual information contained in email is extracted and used for the purpose of discrimination.

Motivation and Background

Spam has become the bane of existence for both Internet users and entities providing email services. Time is lost when sifting through unwanted messages and important emails may be lost through omission or accidental deletion. According to various statistics, spam constitutes the majority of emails sent today and a large portion of emails actually delivered. This translates to large costs related to bandwidth and storage use. Spam detection systems help to alleviate these issues, but they may introduce problems of their own, such as more complex user interfaces, delayed message delivery, and accidental filtering of legitimate messages. It is not clear if any one approach to fighting spam can lead to its complete eradication and a multitude of approaches have been proposed and implemented. Among existing techniques are those relying on the use of supervised and unsupervised machine learning techniques, which aim to derive a model differentiating spam from legitimate content using textual and nontextual attributes. These methods have become an important component of the antispam arsenal and draw from the body of related research such as text classification, fraud detection and cost-sensitive learning. The text mining component of these techniques is of particular prominence given that email messages are primarily composed of text. Application of machine learning and data mining to the spam domain is challenging, however, due, among others, to the adversarial nature of the problem (Dalvi et al. 2004; Fawcett 2003).

Structure of the Learning System

Overview

A machine-learning approach to spam filtering relies on the acquisition of a learning sample of email data, which is then used to induce a classification or scoring model, followed by tuning and setup to satisfy the desired operating conditions. Domain knowledge may be injected at various stages into the induction process. For example, it is common to a priori specific features that are known be highly correlated with the spam label, e.g., certain patterns contained in email headers or certain words or phrases. Depending on the application environment, messages classified as spam are prevented from being delivered (e.g., are blocked or “bounced”), or are delivered with a mechanism to alert users to their likely spam nature. Filter deployment is followed by continuous evaluation of its performance, often accompanied by the collection of error feedback from its users.

Data Acquisition

A spam filtering system relies on the presence of labeled training data, which are used to induce a model of what constitutes spam and what is legitimate email. Spam detection represents a two-class problem, although it may sometimes be desired to introduce special handling of messages for which a confident decision, either way, cannot be made. Depending on the application environment, the training data may represent emails received by one individual or a group of users. Ideally, the data should correspond to a uniform sample acquired over some period of time preceding filter deployment. Typical problems with data collection revolve around privacy issues, whereby users are unwilling to donate emails of personal or sensitive nature. Additionally, lab- eling mistakes are common where legitimate emails may be erroneously marked as spam or vice versa. Also, since for certain types of emails, the spam/legitimate distinction is personal, one may find that the same message content is labeled in a conflicting manner by different users (or even by the same user at different times). Therefore, data cleaning and conflict resolution techniques may need to be deployed, especially when building filters that serve a large and diverse user population.

Due to privacy concerns, few large publicly email corpora exist. The ones created for the TREC Spam Track (TREC data is available from: http://plg.uwaterloo.ca/ gvcormac/treccorpus/). stand out in terms of size and availability of published comparative results.

Content Encoding and Deobfuscation

Spam has been evolving in many ways over the course of time. Some changes reflect the shift in content advertised in such messages (e.g., from pornography and pharmaceuticals to stock schemes and *phish*). Others reflect the formatting of content. While early spam was sent in the form of plain text, it subsequently evolved into more complex HTML, with deliberate attempts to make extraction of meaningful textual features as difficult as possible. Typically, obfuscation (a list of obfuscation techniques is maintained at http://www.jgc.org/tsc.html) aims at

(a) Altering the text extracted from the message for words visible to the user (e.g., by breaking up the characters in message source by HTML tags, encoding the characters in various ways, using character look-alikes, wrapping the display of text using script code executed by the message viewer). This tactic is used to hide the message "payload."
(b) Adding content that is not visible to the user (e.g., using the background color or zero-width font to render certain characters/words). This tactic typically attempts to add "legitimate content."
(c) Purposeful misspelling of words known to be fairly incriminating (e.g., *Viagra* as *V1agr@*), in a way that allows the email recipient to still understand the spammer's message.

The line of detection countermeasures aiming at preventing effective content extraction continues in the form of image spam, where the payload message is encoded in the form of an image that is easily legible to a human but poses challenges to an automatic content extraction system. To the extent that rich and multimedia content gets sent out by legitimate users in increasing proportions, spammers are likely to use the complexity of these media to obfuscate their messages even further. The very fact that obfuscation is attempted, however, provides an opportunity for machine learning techniques to use obfuscation presence as a feature. Thus, even if payload content cannot be faithfully decoded, the very presence of elaborate encoding may help in identifying spam.

Feature Extraction and Selection

An email message represents a semistructured document, commonly following the rfc822 standard (www.faqs.org/rfcs/rfc822.html). Its header consists of fields indicative of formatting, authorship, and delivery information, while its body contains the actual content being sent. There can be little correctness enforcement of the header fields and spamming techniques often rely on spoofing and forging of the header data, although this may provide evidence of tempering. Many early approaches to detect spam depended predominantly on hand-crafted rules identifying inconsistencies and peculiarities of spam email headers. Manually or automatically generated header features continue to be relevant even when other features (e.g., message text) are considered.

Given that an email message tends to be primarily text, features traditionally useful in text categorization have also been found useful in spam detection. These include individual words, phrases, character n-grams, and other textual components (Siefkes et al. 2004). Natural language processing (NLP) techniques such as stemming, stop-word removal, and case folding are also sometimes applied to normalize the features further. Text extraction is often nontrivial due to the application of content obfuscation techniques. For example, standard lexical feature extractors may need to be strengthened to correctly identify word boundaries (e.g., in cases where groups of characters within a word are separated by zero-width HTML tags).

Extraction of features from nontextual attachments (e.g., images, audio, and video) is also possible but tends to be more computationally demanding. Other types of features capture the way a message if formatted, encoded in HTML, composed of multiple parts, etc.

Although nontextual features have different properties than text, it is common practice to combine them with textual features and present a single unified representation to the classifier. Indeed, some approaches make no distinction between text and formatting even during the process of feature extraction, and apply pattern discovery techniques to identifying complex features automatically (Rigoutsos and Huynh 2004). The advantage of such techniques is that they do not require rich domain knowledge and can discover new useful patterns. Due to the large space of possible patterns they can potentially be computationally expensive. However, even the seemingly simplistic treatment of an email message as a plain-text document with "words" delimited by white space often leads to very good results.

Even though typical text documents are already very sparse, the problem is even more pronounced for the email medium due to frequent misspelling and deliberate randomization performed by spammers. Insisting on using all such variations may lead to overfitting for some classifiers, and it leads to large filter memory footprints that are undesirable from an operational standpoint. However, due to the constantly changing distribution of content, it may be dangerous to rely on very few features. Traditional approaches to feature selection based on measures such as Information Gain have been reported as successful in the spam filtering domain, but even simple rudimentary attribute selection based on removing very rare and/or very frequent features tends to work well.

There are a number of entities that can be extracted from message text and that tend to be of relevance in spam detection. Among others, there are telephone numbers and URLs. In commercial email and in spam, these provide a means of ordering products and services and thus, offer important information for vendor and campaign tracking. Detection of signature and mailing address blocks can also be of interest, even if only to indicate their presence or absence.

Learning Algorithms

A variety of learning algorithms have been applied in the spam filtering domain. These range from linear classifiers such as Naive Bayes (Metsis et al. 2006), logistic regression (Goodman and Yih 2006), or linear support vector machines (Drucker et al. 1999; Kołcz and Alspector 2001; Sculley and Wachman 2007) to nonlinear ones such as boosted decision trees (Carreras and Màrquez 2001). Language modeling and statistical compression techniques have also been found quite effective (Bratko et al. 2006). In general, due to the high dimensionality of the feature space, the classifier chosen should be able to handle tens of thousand, or more, attributes without overfitting the training data.

It is usually required that the learned model provides a scoring function, such that for email message x score(x) $\in$ R, with higher score values corresponding to higher probability of the message being spam. The score function can also be calibrated to represent the posterior probability Pspam $|x \in 0, 1$, although accurate calibration is difficult due to constantly changing class and content distributions. The scoring function is used to establish a decision rule:

$$\text{score}(x) \geq th \rightarrow \text{spam}$$

where the choice of the decision threshold th is driven by the minimization of the expected cost. In the linear case, the scoring function takes the form

$$\text{score}(x) = w \cdot x + b$$

where w is the weight vectors, and x is a vector representation of the message. Sometimes scores are normalized with a monotonic function, e.g., to give an estimate of the probability of x being spam.

Linear classifiers tend to provide sufficiently high accuracy, which is also consistent with other application domains involving the text medium. In particular, many variants of the relatively simple Naive Bayes classifier have been found successful in detecting spam, and Naive Bayes often provides a baseline for systems employing more complex classification algorithms (Metsis et al. 2006).

One Model Versus Multiple Models

It often pays off to combine different types of classifiers (even different linear ones) in a sequential or parallel fashion to benefit from the fact that different classifiers may provide an advantage in different regions of the feature space. Stacking via ▶ linear regression has been reported to be effective for this purpose (Sakkis et al. 2001; Segal et al. 2004). One can generally distinguish between cases where all classifiers are induced over the same data and cases where several different datasets are used. In the former case, the combination process exploits the biases of different learning algorithms. In the latter case, one can consider building a multitude of detectors, each targeting a different subclass of spam (e.g., phish, pharmaceutical spam, “Nigerian” scams, etc.). Datasets can also be defined on a temporal basis, so that different classifiers have shorter or longer memory spans. Other criteria of providing different datasets are also possible (e.g., based on the language of the message).

Additional levels of complexity in the classifier combination process can be introduced by considering alternative feature representations for each dataset. For example, a single data collection and a single learning method can be used to create several different classifiers, based upon alternative representations of the same data (e.g., using just the header features or just the message text features).

The method of classifier combination will necessarily depend on their performance and intended area of applications. The combination regimes can range from simple logical-OR through linear combinations to complex nonlinear rules, either derived automatically to maximize the desired performance or specified manually with the guidance of expert domain knowledge.

Off-Line Adaptation Versus Online Adaptation

A spam filtering system can be configured to receive instant feedback from its users, informing it whenever certain messages get misdelivered (this necessarily does not include cases where misclassified legitimate messages are simply blocked). In the case of online filters, the feedback information may be immediately used to update the filtering profile. This allows a filter to adjust to the changing distribution of email content and to detection countermeasures employed by spammers. Not all classifiers are easily amenable to the online learning update, although online versions of well-known learners such as logistic regression (Goodman and Yih 2006) and linear SVMs (Sculley and Wachman 2007) have been proposed. The distinguishing factor is the amount of the original training data that needs to be retained in addition to the model itself to perform future updates. In this respect, Naive Bayes is particularly attractive since it does not require any of the original data for adaptation, with the model itself providing all the necessary information.

One issue with the user feedback signal, however, is its bias toward current errors of the classifier, which for learners depending on the training data being an unbiased sample drawn from the underlying distribution may lead to overcompensation rather than an improvement in filtering accuracy. As an alternative, unbiased feedback can be obtained by either selectively querying users about the nature of uniformly sampled messages or by deriving the labels implicitly.

In the case where off-line adaptation is in use, the feedback data is collected and saved for later use, whereby the filtering models are retrained periodically or only as needed using the data collected. The advantage of off-line adaptation is that it offers more flexibility in terms of the learning algorithm and its optimization. In particular, model retraining can take advantage of a larger quantity of data, and does not have to be constrained to be an extension of the current version of the model. As a result, it is, e.g., possible to redefine the features from one version of the spam filter to the next. One disadvantage is that model updates are likely to be performed less frequently and may be lagging behind the most recent spam trends.

User-Specific Versus User-Independent Spam Detection

What constitutes a spam message tends to be personal, at least for some types of spam. Various commercial messages, such as promotions and advertisements, e.g., may be distributed in a solicited or unsolicited manner, and sometimes only the end recipient may be able to judge which. In that sense, user-specific spam detection has the potential of being most accurate, since a user’s own judgments are used to drive the training process. Since the nonspam content received by any particular user is likely to be more narrowly distributed when compared a larger user population, this makes the discrimination problem much simpler. Additionally, in the adversarial context, a spammer should find it more difficult to measure the success of penetrating personalized filter defenses, which makes it more difficult to craft a campaign that reaches sufficiently many mail inboxes to be profitable.

One potential disadvantage of such solutions is the need for acquiring labeled data on a user by user basis, which may be challenging. For some users historical data may not yet exist (or has already been destroyed), for others even if such data exist, labeling may seem too much of a burden for the users. Aside from the data collection issues, personal spam filtering faces maintainability issues, as the filter is inherently controlled by its user. This may result in less-than-perfect performance, e.g., if the user misdirects filter training.

From the perspective of institutions and email service providers, it is often more attractive to maintain just one set of spam filters that service a larger user population. This makes them simpler to operate and maintain, but their accuracy may depend on the context of any particular user. The advantage of centralized filtering when serving large user populations is that global trends can be more readily spotted and any particular user may be automatically protected against spam, affecting other users. Also, the domain knowledge of the spam-filtering analysts can be readily injected into the filtering pipeline.

To the extent that a service provider maintains personal filters for its population of users, there are potential large system costs to account for, so that a complete cost-benefit analysis needs to be performed to assess the suitability of such as solution as opposed to a user-independent filtering complex. More details on the nature of such trade-offs can be found in Kołcz et al. (2006).

Clustering and Volumetric Techniques

Content clustering can serve as an important data understanding technique in spam filtering. For example, large clusters can justify the use of specialized classifiers and/or the use of cost-sensitive approaches in classifier learning and evaluation (e.g., where different costs are assigned to different groups of content within each class (Kołcz and Alspector 2001).

Both spam and legitimate commercial emails are often sent in large campaigns, where the same or highly similar content is sent to a large number of recipients, sometimes over prolonged periods of time. Detection of email campaigns can therefore play an important role in spam filtering. Since individual messages of a campaign are highly similar to one another, this can be considered a variant of near-replica document detection (Kołcz 2005). It can also be seen as relying on identification of highly localized spikes in the content density distribution. As found in Yoshida et al. (2004), density distribution approaches can be highly effective, which is especially attractive given that they do not require the explicitly labeled training data. Tracking of spam campaigns may be made difficult due to content randomization, and some research has been directed at making the detection methods robust in the presence such countermeasures (Kołcz 2005; Kołcz and Chowdhury 2007).

Misclassification Costs and Filter Evaluation

An important aspect of spam filtering is that the costs of misclassifying spam as legitimate email are not the same as the costs of making the opposite mistake. It is thus commonly assumed that the costs of a false positive mistake (i.e., a legitimate email being misclassified as spam) are much higher than the cost of mistaking spam for legitimate email. Given the prevalence of spam π and the false-spam (FS) and false-legitimate (FL) rates of the classifier, the misclassification cost c can be expressed as

$$c = C_{\mathrm{FS}} \cdot (1 - \pi) \cdot \mathrm{FS} + C_{\mathrm{FL}} \cdot \pi \cdot \mathrm{FL}$$

where C_{FS} and C_{FL} are the costs of making a false-spam and false-legitimate mistake, respectively (there is no penalty for making the correct decision). Since actual values of C_{FS} and C_{FL} are difficult to quantify, one typically sees them combined in the form of a ratio, $\lambda = C_{\mathrm{FS}}/C_{\mathrm{FL}}$, and the overall cost can be expressed as relative to the cost of a false-legitimate misclassification e.g.,

$$c_{\mathrm{rel}} = \lambda \cdot (1 - \pi) \cdot \mathrm{FS} + \pi \cdot FL$$

Practical choices of λ tend to range from 1 to 1,000. Nonuniform misclassification costs can be used during the process of model induction or in postprocessing when setting up the operating parameters of a spam filter, e.g., using the receiver operating characteristic (ROC) analysis.

Since the costs and cost ratios are sometimes hard to define, some approaches to evaluation favor direct values of the false-spam and false-legitimate error rates. This captures the intuitive requirement that an effective spam filter should provide high detection rate at a close-to-zero false-spam rate. Alternatively, threshold independent metrics such as the area under the ROC (AUC) can be used (Bratko et al. 2006; Cormack and Lynam 2006), although other measures have also been proposed (Sakkis et al. 2001).

Adaptation to Countermeasures

Spam filtering is an inherently adversarial task, where any solution deployed on a large scale is likely to be met with a response on the part of the spammers. To that extent that the success of a spam filter can be pinpointed to any particular component (e.g., the type of features used), that prominent component is likely to be attacked directly and may become a victim of its own success. For example, the use of word features in spam filtering encourages countermeasures in the form of deliberate misspellings, word fragmentation, and "invisible ink" in HTML documents. Also, since some words are considered by a model inherently more legitimate than others, "word stuffing" has been used to inject large blocks of potentially legitimate vocabulary into an otherwise spammy message in the hope that this information outweighs the evidence provided by the spam content (Lowd and Meek 2005).

Some authors have attempted to put the adversarial nature of spam filtering in the formal context of game theory (Dalvi et al. 2004). One difficulty of drawing broad conclusion based on such analyses is the breadth of the potential attack/defense front, of which only small sections have been successfully captured in the game-theory formalism. The research on countering the countermeasures points at using multiple diverse filtering components, normalization of features to keep them invariant to irrelevant alterations. A key point is that frequent filter retraining is likely to help in keeping up with the shifts in content distribution, both natural and due to countermeasures.

Future Directions

Reputation Systems and Social Networks

There has been a growing interest in developing reputation systems capturing the trustworthiness of a sender with respect to a particular user or group of users. To this end however, the identity of the sender needs to be reliably verified, which poses challenges and presents a target for potential abuses of such systems. Nevertheless, reputation systems are likely to grow in importance, since they are intuitive from the user perspective in capturing the communication relationships between users. Sender reputation can be hard or soft. In the hard variant, the recipient always accepts or declines messages from a given sender. In the soft variant, the reputation reflects the level of trustworthiness of the sender in the context of the given recipient. When sender identities resolve to individual email addresses, the reputation system can be learned via analysis of a large social network that documents who exchanges email with whom. The sender identities can also be broader however, e.g., assigning reputation to a particular mail server or all mail servers responsible for handling the outbound traffic for a particular domain. On the recipient side, reputation can also be understood globally to represent the trustworthiness of the sender with respect to all recipients hosted by the system. Many open questions remain with regard to computing and maintaining reputations as well as using them effectively to improve spam detection. In the context of text mining, one such question is the extent to which email content analysis can be used to aid the process of reputation assessment.

Cross-References

- ▸ Cost-Sensitive Learning
- ▸ Document Categorization

- ▶ Linear Separability
- ▶ Logistic Regression
- ▶ Naïve Bayes
- ▶ Support Vector Machines

Recommended Reading

Bratko A, Cormack GV, Filipic B, Lynam TR, Zupan B (2006) Spam filtering using statistical data compression models. J Mach Learn Res 7:2673–2698

Carreras X, Màrquez L (2001) Boosting trees for anti-spam email filtering. In: Proceedings of RANLP-01, the 4th international conference on recent advances in natural language processing. ACM, New York

Cormack GV, Lynam TR (2006) On-line supervised spam filter evaluation. ACM Trans Inf Syst 25(3):11

Dalvi N, Domingos P, Sanghai MS, Verma D (2004) Adversarial classification. In: Proceedings of the tenth international conference on knowledge discovery and data mining, vol 1. ACM, New York, pp 99–108

Drucker H, Wu D, Vapnik VN (1999) Support vector machines for spam categorization. IEEE Trans Neural Netw 5(10):1048–1054

Fawcett T (2003) In vivo' spam filtering: a challenge problem for data mining. KDD Explor 5(2):140–148

Goodman J, Yih W (2006) Online discriminative spam filter training. In: Proceedings of the third conference on email and anti-spam (CEAS-2006), Mountain View

Kołcz A (2005) Local sparsity control for naive bayes with extreme misclassification costs. In: Proceedings of the eleventh ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York

Kołcz A, Alspector J (2001) SVM-based filtering of e-mail spam with content-specific misclassification costs. In: TextDM'2001 (IEEE ICDM-2001 workshop on text mining), San Jose

Kołcz A, Bond M, Sargent J (2006) The challenges of service-side personalized spam filtering: scalability and beyond. In: Proceedings of the first international conference on scalable information systems (INFOSCALE). ACM, New York

Kołcz AM, Chowdhury A (2007) Hardening fingerprinting by context. In: Proceedings of the fourth international conference on email and anti-spam, Mountain View

Lowd D, Meek C (2005) Good word attacks on statistical spam filters. In: Proceedings of the second conference on email and anti-spam (CEAS-2005), Mountain View

Metsis V, Androutsopoulos I, Paliouras G (2006) Spam filtering with naive bayes – which naive bayes? In: Proceedings of the third conference on email and anti-spam (CEAS-2006), Mountain View

Rigoutsos I, Huynh T (2004) Chung-Kwei: a pattern-discovery-based system for the automatic identification of unsolicited e-mail messages (SPAM). In: Proceedings of the first conference on email and anti-spam (CEAS-2004), Mountain View

Sahami M, Dumais S, Heckerman D, Horvitz E (1998) A Bayesian approach to filtering junk email. In: AAAI workshop on learning for text categorization, Madison. AAAI technical report WS-98-05

Sakkis G, Androutsopoulos I, Paliouras G, Karkaletsis V, Spyropoulos CD, Stamatopoulos P (2001) Stacking classifiers for anti-spam filtering of e-mail. In: Lee L, Harman D (eds) Proceedings of empirical methods in natural language processing (EMNLP 2001), pp 44–50. http://www.cs.cornell.edu/home/llee/emnlp/proceeding.html

Sculley D, Wachman G (2007) Relaxed online support vector machines for spam filtering. In: Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval. ACM, New York

Segal R, Crawford J, Kephart J, Leiba B (2004) SpamGuru: an enterprise anti-spam filtering system. In: Proceedings of the first conference on email and anti-spam (CEAS-2004), Mountain View

Siefkes C, Assis F, Chhabra S, Yerazunis W (2004) Combining winnow and orthogonal sparse bigrams for incremental spam filtering. In: Proceedings of the European conference on principle and practice of knowledge discovery in databases. Springer, New York

Yoshida K, Adachi F, Washio T, Motoda H, Homma T, Nakashima A et al (2004) Densitiy-based spam detection. In: Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, pp 486–493

Text Mining for the Semantic Web

Marko Grobelnik[1], Dunja Mladenić[1], and Michael Witbrock[2]

[1]Artificial Intelligence Laboratory, Jožef Stefan Insitute, Ljubljana, Slovenia

[2]Cycorp Inc, Austin, TX, USA

Definition

▶ Text Mining methods allow for the incorporation of textual data within applications of semantic technologies on the Web. Application of these techniques is appropriate when some of the data needed for a Semantic Web use scenario are in textual form. The techniques range from simple processing of text to reducing vocabulary size, through applying shallow ▶ natural language processing to constructing new semantic features or applying information retrieval to selecting relevant texts for analysis, through complex methods involving integrated visualization of semantic information, semantic search, semiautomatic ontology construction, and large-scale reasoning.

Motivation and Background

Semantic Web applications usually involve deep structured knowledge integrated by means of some kind of ontology. Text mining methods, on the other hand, support the discovery of structure in data and effectively support semantic technologies on data-driven tasks such as (semi)automatic ▶ ontology acquisition, extension, and mapping. Fully automatic text mining approaches are not always the most appropriate for combination with Semantic Web content, because often it is too difficult or too costly to fully integrate the available background domain knowledge into a suitable representation. For such cases, semiautomatic methods, such as ▶ Active Learning and ▶ Semi-supervised Text Processing (see ▶ Semi-supervised Learning), can be applied to make use of small pieces of human knowledge to provide guidance toward the desired ontology or other models. Application of these semiautomated techniques can reduce the amount of human effort required to produce training data by an order of magnitude while preserving the quality of results.

To date, Semantic Web applications have typically been associated with data, such as text documents, and corresponding metadata that have been designed to be relatively easily manageable by humans. Humans are, for example, very good at reading and understanding text and tables. General semantic technologies, on the other hand, aim more broadly at handling data modalities including multimedia, signals from emplaced or remote sensors, and the structure and content of communication and transportation graphs and networks. In handling such multimodal data, much of which is not readily comprehensible by unaugmented humans, there must be significant emphasis on fully or semiautomatic methods offered by knowledge discovery technologies whose application is not limited to a specific data representation (Grobelnik and Mladenic 2005).

Data and the corresponding semantic structures change over time, and semantic technologies also aim at adapting the ontologies that model the data accordingly. For most such scenarios, extensive human involvement in building models and adapting them according to the data is too costly, too inaccurate, and too slow. ▶ Stream mining (Gaber et al. 2005) techniques (Data Streams: Clustering) allow text mining of dynamic data (e.g., notably in handling a stream of news or of public commentary).

Ontology is a fundamental method for organizing knowledge in a structured way and is applied, along with formalized reasoning, in areas from philosophy to scientific discovery to knowledge management and the Semantic Web. In computer science, an ontology generally refers to a graph or network structure consisting of a set of concepts (vertices in a graph), a set of relationships connecting those concepts (directed edges in a graph), and, possibly, a set of distinguished instance concepts assigned to particular class concepts (data records assigned to vertices in a graph). Although much useful knowledge can be represented by the ground binary relations most conveniently encoded as graphs, more complex relationships involving more than two entities are needed, and the graph metaphor is more remote. In many cases, knowledge is structured in one of these ways to allow for automated inference based on a logical formalism such as the ▶ predicate calculus (Barwise and Etchemendy 2002); for these applications, an ontology often further comprises a set of rules or produces new knowledge within the representation from existing knowledge. An ontology containing both instance data and rules for its application is often referred to as a knowledge base (KB) (e.g., Lenat 1995).

Machine learning practitioners refer to the task of automatically constructing these ontologies as ▶ ontology learning. From this point of view, an ontology is seen as a class of models – somewhat more complex than most used in machine learning – which need to be expressed in some ▶ Hypothesis Language. This definition of ontology learning (from Grobelnik and Mladenic 2005) enables a decomposition into several machine learning tasks, including ▶ learning concepts, identifying relationships between existing concepts, populating an existing ontology/structure with instances, identifying change in dynamic ontologies, and inducing rules over concepts, background knowledge, and instances.

Following this scheme, text mining methods have been applied to extending existing ontologies based on Web documents, learning semantic relations from text based on collocations, semiautomatic data-driven ontology construction based on ▶ document clustering and classification, extracting semantic graphs from text, transforming text into ▶ RDF triples (a commonly used Semantic Web data representation), and mapping triplets to semantic classes using several kinds of lexical and ontological background knowledge. Text mining is also intensively used in the effort to produce a Semantic Web for annotation of text with concepts from ontology. For instance, a text document is split into sentences, each sentence is represented as a word vector, sentences are clustered, and each cluster is labeled by the most characteristic words from its sentences and mapped upon the concepts of a general ontology. Several approaches that integrate ontology management, knowledge discovery, and human language technologies are described in Davies et al. (2009).

Extending the text mining paradigm, efforts are aimed at giving machines an approximation of the full human ability to acquire knowledge from text. Some of the systems (Curtis et al. 2009; Mitchell 2005; Rusu 2014) actively use background knowledge in the extraction process for disambiguation or knowledge structuring. Machine reading aims at full text understanding by integrating knowledge-based construction and use into syntactically sophisticated natural ▶ language analysis, leading to systems that autonomously improve their ability to extract further knowledge from text (e.g., Curtis et al. 2009; Etzioni et al. 2007; Mitchell 2005; Starc and Fortuna 2012; Starc and Mladenic 2013).

Biomedical Text Mining

Because of the development and widespread use of high-quality biomedical knowledge bases, such as the Gene Ontology (Ashburner et al. 2000), Cell Ontology (Bard et al. 2005), and Linked Neuron Data (Zeng et al. 2015), and the overwhelming volume of the relevant literature (24 million biomedicine citations in PubMed), biomedical knowledge extraction is subject to a great deal of research. Relevant shared evaluation tasks include BioCreative (Hirschman et al. 2005) and BioNLP (Cohen et al. 2014). Although much of the work on biological fact extraction still relies on supervised training with closely annotated training data, with the risk of over-constraining the mapping of semantics to particular text substrings, volume of high-quality Semantic Web fact bases has enabled more natural training methods, such as distant supervision (Augenstein et al. 2014).

Cross-References

▶ Active Learning
▶ Classification
▶ Clustering
▶ Semi-supervised Learning
▶ Semi-supervised Text Processing
▶ Text Mining
▶ Text Visualization

Recommended Reading

Ashburner M, Ball CA, Blake JA, Botstein D, Butler H, Cherry JM, Davis AP, Dolinski K, Dwight SS, Eppig JT, Harris MA, Hill DP, Issel-Tarver L, Kasarskis A, Lewis S, Matese JC, Richardson JE, Ringwald M, Rubin GM, Sherlock G (2000) Gene ontology: tool for the unification of biology. Nat Genet 25(1):25–29

Augenstein I, Maynard D, Ciravegna F (2014) Relation extraction from the web using distant supervision.

In: Janowicz K et al (eds) EKAW 2014. LNAI 8876. Springer, pp 26–41

Bard J, Rhee SY, Ashburner M (2005) An ontology for cell types. Genome Biol 6(2):R21

Barwise J, Etchemendy J (2002) Language proof and logic. Center for the study of language and information. ISBN:157586374X

Buitelaar P, Cimiano P, Magnini B (2005) Ontology learning from text: methods, applications and evaluation, frontiers in artificial intelligence and applications. IOS Press, Amsterdam

Cohen K, Demner-Fushman D, Ananiadou S, Tsujii J-i (2014) Proceedings of BioNLP 2014, Baltimore. Association for Computational Linguistics

Curtis J, Baxter D, Wagner P, Cabral J, Schneider D, Witbrock M (2009) Methods of rule acquisition in the TextLearner system. In: Proceedings of the 2009 AAAI spring symposium on learning by reading and learning to read. AAAI Press, Palo Alto, pp 22–28

Davies J, Grobelnik M, Mladenić D (2009) Semantic knowledge management. Springer, Berlin

Etzioni O, Banko M, Cafarella MJ (2007) Machine reading. In: Proceedings of the 2007 AAAI spring symposium on machine reading

Gaber MM, Zaslavsky A, Krishnaswamy S (2005) Mining data streams: a review. ACM SIGMOD Rec 34(1):18–26. ISSN:0163-580

Grobelnik M, Mladenic D (2005) Automated knowledge discovery in advanced knowledge management. J Knowl Manag 9:132–149

Hirschman L, Yeh A, Blaschke C, Valencia A (2005) Overview of BioCreAtIvE: critical assessment of information extraction for biology. BMC Bioinform 6(Suppl 1):S1

Lenat DB (1995) Cyc: a large-scale investment in knowledge infrastructure. Commun ACM 38(11):33–38

Mitchell T (2005) Reading the web: a breakthrough goal for AI. Celebrating twenty-five years of AAAI: notes from the AAAI-05 and IAAI-05 conferences. AI Mag 26(3):12–16

Rusu D (2014) Text annotation using background knowledge. Doctoral Dissertation, Jozef Stefan International Postgraduate School, Ljubljana

Starc J, Fortuna B (2012) Identifying good patterns for relation extraction. In: Proceedings of the 15th international multiconference information society – IS 2012. Institut Jožef Stefan, Ljubljana, pp 205–208

Starc J, Mladenic D (2013) Semi-automatic construction of pattern rules for translation of natural language into semantic representation. In: Proceedings of the 5th Jožef Stefan International Postgraduate School Students Conference, Jožefa Stefana International Postgraduate School, pp 199–208

Zeng Y, Wang D, Zhang T Linked brain data. Web http://www.linked-neuron-data.org/. Retrieved 11 Jan 2015

Text Spatialization

▶ Text Visualization

Text Visualization

John Risch[1], Shawn Bohn[1], Steve Poteet[2], Anne Kao[2], Lesley Quach[2], and Jason Wu[2]
[1]Pacific Northwest National Laboratory, Richland, WA, USA
[2]Boeing Phantom Works, Seattle, WA, USA

Synonyms

Semantic mapping; Text spatialization; Topic mapping

Definition

The term *text visualization* describes a class of knowledge discovery techniques that use interactive graphical representations of textual data to enable knowledge discovery via recruitment of human visual pattern recognition and spatial reasoning capabilities. It is a subclass of *information visualization*, which more generally encompasses visualization of nonphysically based (or "abstract") data of all types. Text visualization is distinguished by its focus on the unstructured (or *free text*) component of information. While the term "text visualization" has been used to describe a variety of graphical methods for deriving knowledge from text, it is most closely associated with techniques for depicting the semantic characteristics of large document collections. Text visualization systems commonly employ unsupervised machine learning techniques as part of broader strategies for organizing and graphically representing such collections.

Motivation and Background

The Internet enables universal access to vast quantities of information, most of which (despite admirable efforts Berners-Lee et al. 2001) exists in the form of unstructured and unorganized text. Advancements in search technology make it possible to retrieve large quantities of this information with reasonable precision; however, only a tiny fraction of the information available on any given topic can be effectively exploited.

Text visualization technologies, as forms of computer-supported knowledge discovery, aim to improve our ability to understand and utilize the wealth of text-based information available to us. While the term "text visualization" has been used to describe a variety of techniques for graphically depicting the characteristics of free-text data (Havre et al. 2002; Small 1996), it is most closely associated with the so-called *semantic clustering* or *semantic mapping* techniques (Chalmers and Chitson 1992; Kohonen et al. 2000; Lin et al. 1991; Wise et al. 1995). These methods attempt to generate summary representations of document collections that convey information about their general topical content and similarity structure, facilitating general domain understanding and analytical reasoning processes.

Text visualization methods are generally based on vector space models of text collections (Salton 1989), which are commonly used in information retrieval, clustering, and categorization. Such models represent the text content of individual documents in the form of vectors of frequencies of the terms (*text features*) they contain. A document collection is therefore represented as a collection of vectors. Because the number of unique terms present in a document collection generally is in the range of tens of thousands, a dimensionality reduction method such as singular value decomposition (SVD) (Deerwester et al. 1990) or other matrix decomposition method (Kao et al. 2008; Booker et al. 1999) is typically used to eliminate noise terms and reduce the length of the document vectors to a tractable size (e.g., 50–250 dimensions). Some systems attempt to first identify discriminating features in the text and then use mutual probabilities to specify the vector space (York et al. 1995).

To enable visualization, the dimensions must be further reduced to two or three. The goal is a graphical representation that employs a "spatial proximity means conceptual similarity" metaphor where topically similar text documents are represented as nearby points in the display. Various regions of the semantic map are subsequently labeled with descriptive terms that convey the primary concepts described by nearby documents. The text visualization can thus serve as a kind of graphical "table of contents" depicting the conceptual similarity structure of the collection.

Text visualization systems therefore typically implement four key functional components, namely,

1. A *tokenization* component that characterizes the lexical content of text units via extraction, normalization, and selection of key terms
2. A *vector space modeling* component that generates a computationally tractable vector space representation of a collection of text units
3. A *spatialization* component that uses the vector space model to generate a 2D or 3D spatial configuration that places the points representing conceptually similar text units in near spatial proximity
4. A *labeling* component that assigns characteristic text labels to various regions of the semantic map

Although machine learning techniques can be used in several of these steps, their primary usage is in the spatialization stage. An unsupervised learning algorithm is typically used to find meaningful low-dimensional structures hidden in high-dimensional document feature spaces.

Structure of Learning System

Spatialization is a term generically used in ▸information visualization to describe the process of generating a spatial representation of inherently nonspatial information. In the context of text visualization, this term generally refers to the application of a nonlinear dimensionality reduction algorithm to a collection of text vectors in order to generate a visually interpretable two- or three-dimensional representation of the similarity structure of the collection. The goal is the creation of a *semantic similarity map* that positions graphical features representing text units (e.g., documents) conceptually similar to one another near one another in the visualization display. These maps may be further abstracted to produce more general summary representations of text collections that do not explicitly depict the individual text units themselves (Wise et al. 1995).

A key assumption in text visualization is that text units which express similar concepts will employ similar word patterns and that the existence of these word correlations creates coherent structures in high-dimensional text feature spaces. A further assumption is that text feature spaces are nonlinear but that their structural characteristics can be approximated by a smoothly varying low-dimensional manifold. The text spatialization problem thus becomes one of finding an embedding of the feature vectors in a two- or three-dimensional manifold that best approximates this structure. Because the intrinsic dimensionality of the data is invariably much larger than two (or three), significant distortion is unavoidable. However, because the goal of text visualization is not necessarily the development of an accurate representation of interdocument similarities, but rather the depiction of broad (and ambiguously defined) semantic relationships, this distortion is generally considered acceptable.

Text vector spatialization therefore involves the fitting of a model into a collection of observations. Most text visualization systems developed to date have employed some type of unsupervised learning algorithm for this purpose. In general, the desired characteristics of an algorithm used for text spatialization include that it (1) preserves global properties of the input space, (2) preserves the pairwise input distances to the greatest extent possible, (3) supports out-of-sample extension (i.e., the incremental addition of new documents), and (4) has low computational and memory complexity. Computational and memory costs are key considerations, as a primary goal of text visualization is the management and interpretation of extremely large quantities of textual information.

A leading approach is to iteratively adapt the nodes of a fixed topology mesh to the high-dimensional feature space via adaptive refinement. This is the basis of the well-known Kohonen feature mapping algorithm, more commonly referred to as the ▶ self-organizing map (SOM) (Kohonen 1997). In a competitive learning process, text vectors are presented one at a time to a (typically triangular) grid, the nodes of which have been randomly initialized to points in the term space. The Euclidean distance to each node is computed, and the node closest to the sample is identified. The position of the winning node, along with those of its topologically nearest neighbors, is incrementally adjusted toward the sample vector. The magnitude of the adjustments is gradually decreased over time. The process is generally repeated using every vector in the set for many hundreds or thousands of cycles until the mesh converges on a solution. At the conclusion, the samples are assigned to their nearest nodes, and the results are presented as a uniform grid. In the final step, the nodes of the grid are labeled with summary terms which describe the key concepts associated with the text units that have been assigned to them.

Although self-organizing maps can be considered primarily a clustering technique, the grid itself theoretically preserves the topological properties of the input feature space. As a consequence, samples that are nearest neighbors in the feature space generally end up in topologically adjacent nodes. However, while SOMs are topology-preserving, they are not distance-preserving. Vectors that are spatially distant in the input space may be presented as proximal in the output, which may be semantically undesirable. SOMs have a number of attractive characteristics, including straightforward out-of-sample extension and low computational and memory complexity. Examples of the use of SOMs in text visualization applications can be found in Lin et al. (1991), Kaski et al. (1998), and Kohonen et al. (2000).

Often, it is considered desirable to attempt to preserve the distances among the samples in the input space to the greatest extent possible in the output. The rationale is that the spatial proximities of the text vectors capture important and meaningful characteristics of the associated text units: spatial "nearness" corresponds to conceptual "nearness." As a consequence, many text visualization systems employ distance-preserving dimensionality reduction algorithms. By far, the most commonly used among these is the class of algorithms known as *multidimensional scaling* (MDS) algorithms.

Multidimensional scaling is "a term used to describe any procedure which starts with the 'distances' between a set of points (or individuals or objects) and finds a configuration of the points, preferably in a smaller number of dimensions, usually 2 or 3" (Chatfield and Collins 1980, quoted in Chalmers and Chitson 1992). There are two main subclasses of MDS algorithms. Metric (quantitative, also known as classical) MDS algorithms attempt to preserve the pairwise input distances to the greatest extent possible in the output configuration, while nonmetric (qualitative) techniques attempt only to preserve the rank order of the distances. Metric techniques are most commonly employed in text visualization.

Metric MDS maps the points in the input space to the output space while maintaining the pairwise distances among the points to the greatest extent possible. The quality of the mapping is expressed in a stress function which is minimized using any of a variety of optimization methods, e.g., via eigen decomposition of a pairwise dissimilarity matrix, or using iterative techniques such as generalized Newton–Raphson, simulated annealing, or genetic algorithms. A simple example of a stress function is the raw stress function (Kruskal 1964) defined by

$$\phi(Y) = \sum_{ij} (||x_i - x_j|| - ||y_i - y_j||)^2$$

in which $||x_i - x_j||$ is the Euclidean distance between points x_i and x_j in the high-dimensional space and $||y_i - y_j||$ is the distance between the corresponding points in the output space. A variety of alternative stress functions have been proposed (Cox and Cox 2001). In addition to its distance-preserving characteristics, MDS has the added advantage of preserving the global properties of the input space. A major disadvantage of MDS, however, is its high computational complexity, which is approximately $O(kN^2)$, where N is the number of data points and k is the dimensionality of the embedding. Although computationally expensive, MDS can be used practically on data sets of up to several hundred documents in size. Another disadvantage is that out-of-core extension requires reprocessing of the full data set if an optimization method which computes the output coordinates all at once is used.

The popularity of MDS methods has led to the development of a range of strategies for improving on its computational efficiency to enable scaling of the technique to text collections of larger size. One approach is to use either cluster centroids or a randomly sampled subset of input vectors as surrogates for the full set. The surrogates are down-projected independently using MDS, and then the remainder of the data is projected relative to this "framework" using a less expensive algorithm, e.g., distance-based triangulation. This is the basis for the *anchored least stress* algorithm used in the SPIRE text visualization system (York et al. 1995), as well as the more recently developed Landmark MDS (de Silva and Tenenbaum 2003) algorithm.

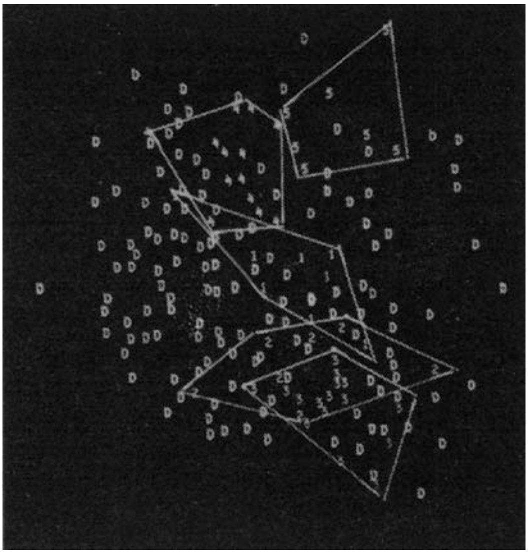
While self-organizing maps and multidimensional scaling techniques have received the most attention to date, a number of other machine learning techniques have also been used for text spatialization. The Starlight system (Risch et al. 1999) uses *stochastic proximity embedding* (Agrafiotis 2003), a high-speed nonlinear manifold learning algorithm. Other approaches have employed methods based on graph layout techniques (Fabrikant 2001). Generally speaking, any of a number of techniques for performing dimensionality reduction in a correlated system of measurements (classified under the rubric of factor analysis in statistics) may be employed for this purpose.

Machine learning algorithms can also be used in text visualization for tasks other than text vector spatialization. For example, generation of descriptive labels for semantic maps requires partitioning of the text units into related sets. Typically, a partitioning-type ▶ clustering algorithm such as K-means is used for this purpose (see ▶ Partitional Clustering), either as an element of the spatialization strategy (see York et al. 1995) or as a postspatialization step. The labeling process itself may also employ machine learning algorithms. For instance, the TRUST system (Booker et al. 1999; Kao et al. 2008) employed by Starlight generates meaningful labels for document clusters using a kind of ▶ unsupervised learning. By projecting a cluster centroid defined in the reduced dimensional representation (e.g., 50–250 dimensions) back into the full-term space, terms related to the content of the documents in the cluster are identified and used as summary terms. Machine learning techniques can also be applied indirectly during the tokenization phase of text visualization. For example, information extraction systems commonly employ rule sets that have been generated by a supervised learning algorithm (Mooney and Bunescu 2006). Such systems may be used to identify tokens that are most characteristic of the overall topic of a text unit or are otherwise of interest (e.g., the names of people or places). In this way, the dimensionality of the input space can be drastically reduced, accelerating downstream processing while simultaneously improving the quality of the resulting visualizations.

Applications

Sammon

The first text visualization system based on a text vector space model was likely a prototype developed in the 1960s by John Sammon's "nonlinear mapping," or *NLM*, algorithm (today referred to as organizing text data). The configuration depicted here is the result of applying Sammon's algorithm to a collection of 188 documents represented as 17-dimensional vectors determined according to document relevance to 1,125 keywords and phrases. Among other interesting and prescient ideas, Sammon describes techniques for interacting with text visualizations depicted on a "CRT display" using a light pen (Fig. 1).



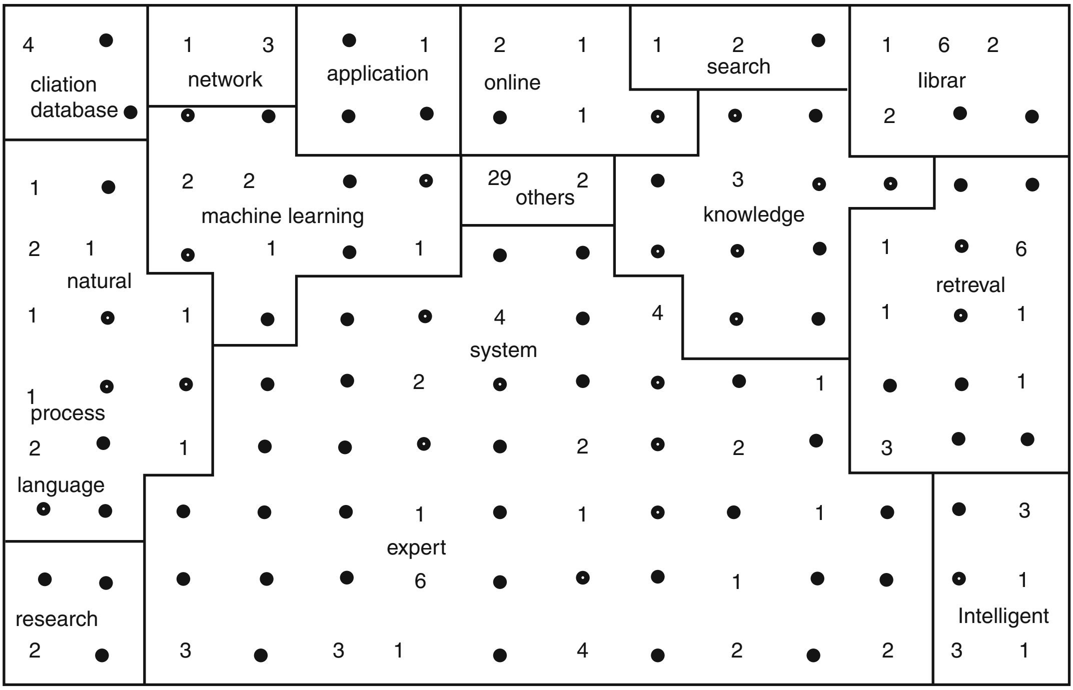
Text Visualization, Fig. 1 Text Visualisation on a CRT display using a light pen

Lin

Lin's 1991 prototype (Lin et al. 1991) was one of the first to demonstrate the use of self-organizing maps for organizing text documents. Lin formed a 25-dimensional vector space model of a 140-document collection using 25 key index terms extracted from the text. The document vectors were used to train a 140-node feature map, generating the result shown here (the fact that the number of nodes matches the number of documents is coincidental). Lin was also among the first to assign text labels to various regions of the resulting map to improve the interpretability and utility of the resulting product (Fig. 2).

BEAD

The BEAD system (Chalmers and Chitson 1992) was a text visualization prototype



Text Visualization, Fig. 2 Labelled self-organising maps

developed during the early 1990s at Rank Xerox EuroPARC. BEAD employed a vector space model constructed using document keywords and a hybrid MDS algorithm based on an optimized form of simulated annealing. Although it did not include a region labeling component, BEAD did support highlighting of visualization features in response to query operations, a now standard text visualization system feature. The BEAD project also pioneered a number of now common interaction techniques and was among the first to explore 3D representations of document collections (Fig. 3).

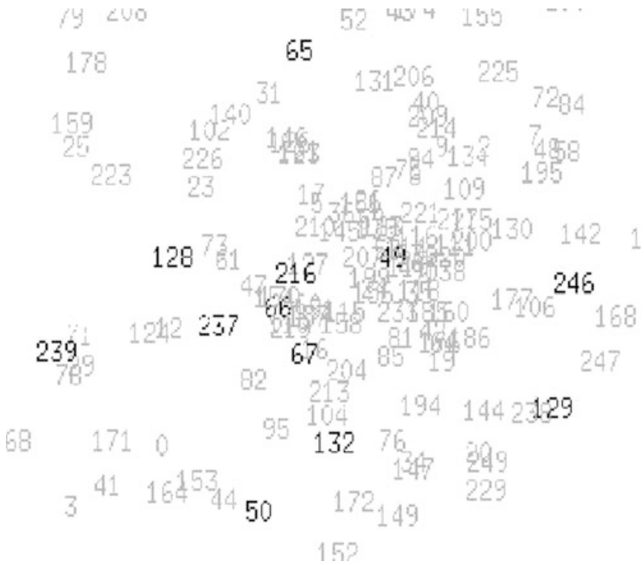
IN-SPIRE

IN-SPIRE (formerly SPIRE, Spatial Paradigm for Information Retrieval and Exploration) (Wise et al. 1995) was originally developed in 1995 at Pacific Northwest National Laboratory (PNNL). Over the years, IN-SPIRE has evolved from using MDS to anchored least stress to a hybrid clustering/PCA projection scheme. The SPIRE/IN-SPIRE system introduced several new concepts, including the use of a 3D "landscape" abstraction (called a *ThemeView*) for depicting the general characteristics of large text collections. A recently developed parallelized version of the software is capable of generating visualizations of document collections containing millions of items (Fig. 4).

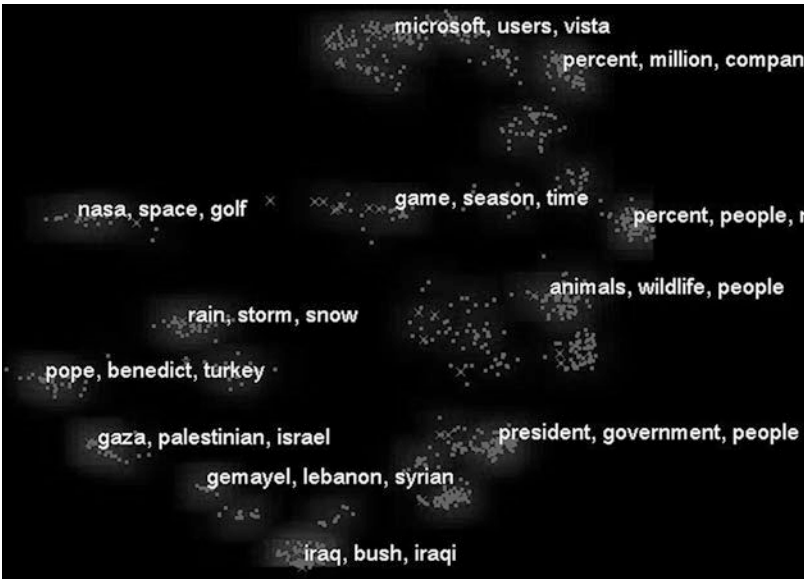
WEBSOM

WEBSOM (Kaski et al. 1998) was another early application of Kohonen self-organizing maps to text data. Early versions of WEBSOM used an independent SOM to generate reduced dimensionality text vectors which were then mapped with a second SOM for visualization purposes. More recent SOM-based text visualization experiments have employed vectors constructed via random projections of weighted word histograms (Kohonen et al. 2000). SOMs have been used to generate semantic maps containing millions of documents (Fig. 5).

Text Visualization, Fig. 3 3D representation of document collections



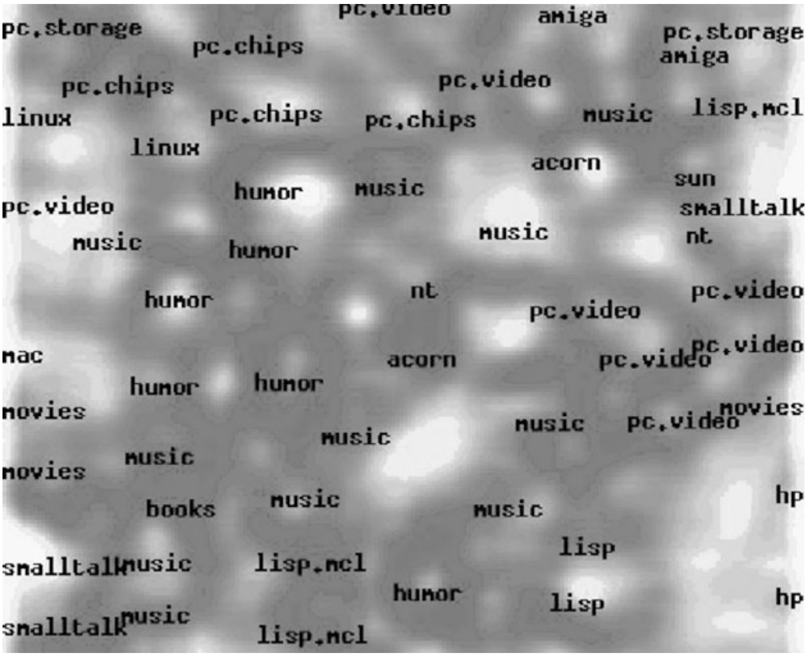
Text Visualization, Fig. 4 Large scale 3D representation of document collections

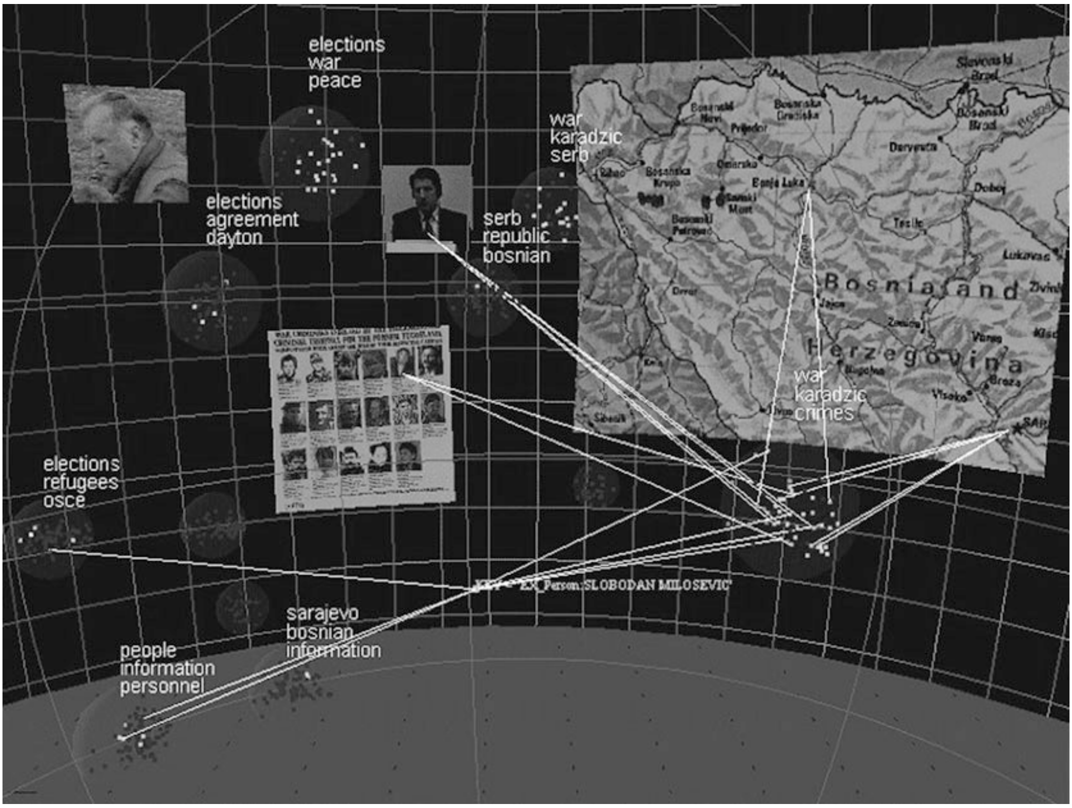


Starlight

Starlight (Risch et al. 1999) is a general-purpose information visualization system developed at PNNL that includes a text visualization component. Starlight's text visualization system uses the Boeing *Text Representation Using Subspace Transformation* (TRUST) text engine for vector space modeling and text summarization. Text vectors generated by TRUST are clustered, and the cluster centroids are down-projected to 2D and 3D using a nonlinear manifold learning algorithm. Individual document vectors associated with each cluster are likewise projected within a local coordinate system established at the projected coordinates of their associated cluster centroid, and TRUST is used to generate topical labels for each cluster. Starlight is unique in that it couples text visualization with a range of other information visualization techniques (such as link displays) to depict multiple aspects of information simultaneously (Fig. 6).

Text Visualization, Fig. 5 Semantic map generated by self-organising maps (SOMs)





Text Visualization, Fig. 6 Starlight link display of multiple aspects of information

Cross-References

- ▶ Data Preprocessing
- ▶ Dimensionality Reduction
- ▶ Document Classification
- ▶ Evolutionary Feature Selection and Construction
- ▶ Self-Organizing Maps
- ▶ Text Visualization

Recommended Reading

Agrafiotis DK (2003) Stochastic proximity embedding. J Comput Chem 24(10):1215–1221

Berners-Lee T, Hendler J, Lassila O (2001) The semantic web. Sci Am 284(5):34–43

Booker A, Condliff M, Greaves M, Holt FB, Kao A, Pierce DJ et al (1999) Visualizing text data sets. Comput Sci Eng 1(4):26–35

Chalmers M, Chitson P (1992) Bead: explorations in information visualization. In: SIGIR '92: proceedings of the 15th annual international ACM SIGIR conference on research and development in information retrieval, Copenhagen. ACM, New York, pp 330–337

Chatfield C, Collins A (1980) Introduction to multivariate analysis. Chapman & Hall, London

Cox MF, Cox MAA (2001) Multidimensional scaling. Chapman & Hall, London

Crouch D (1986) The visual display of information in an information retrieval environment. In: Proceedings of the ACM SIGIR conference on research and development in information retrieval, Pisa. ACM, New York, pp 58–67

Deerwester S, Dumais S, Furnas G, Landauer T, Harshman R (1990) Indexing by latent semantic analysis. J Am Soc Inf Sci 41(6):391–407

de Silva V, Tenenbaum JB (2003) Global versus local methods in nonlinear dimensionality reduction. In: Becker S, Thrun S, Obermayer K (eds) Proceedings of the NIPS, Vancouver, vol 15, pp 721–728

Doyle L (1961) Semantic roadmaps for literature searchers. J Assoc Comput Mach 8(4):367–391

Fabrikant SI (2001) Visualizing region and scale in information spaces. In: Proceedings of the 20th international cartographic conference, ICC 2001, Beijing, pp 2522–2529

Havre S, Hetzler E, Whitney P, Nowell L (2002) ThemeRiver: visualizing thematic changes in large document collections. IEEE Trans Vis Comput Graph 8(1):9–20

Huang S, Ward M, Rundensteiner E (2003) Exploration of dimensionality reduction for text visualization. Technical report TR-03-14, Department of Computer Science, Worcester Polytechnic Institute, Worcester

Kao A, Poteet S, Ferng W, Wu J, Quach L (2008) Latent semantic indexing and beyond, to appear. In: Song M, Wu YF (eds) Handbook of research on text and web mining technologies. Idea Group Inc., Hershey

Kaski S, Honkela T, Lagus K, Kohonen T (1998) WEBSOM-self-organizing maps of document collections. Neurocomputing 21:101–117

Kohonen T (1997) Self-organizing maps. Series in information sciences, vol 30, 2nd edn. Springer, Heidelberg

Kohonen T, Kaski S, Lagus K, Salojärvi J, Honkela J, Paatero V et al (2000) Self organization of a massive document collection. IEEE Trans Neural Netw 11(3):574–585

Kruskal JB (1964) Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. Psychometrika 29(1):1–27

Lin X, Soergel D, Marchionini DA (1991) Self-organizing semantic map for information retrieval. In: Proceedings of the fourteenth annual international ACM/SIGIR conference on research and development in information retrieval, Chicago, pp 262–269

Mooney RJ, Bunescu R (2006) Mining knowledge from text using information extraction. In: Kao K, Poteet S (eds) SIGKDD explorations, pp 3–10

Paulovich FV, Nonato LG, Minghim R (2006) Visual mapping of text collections through a fast high precision projection technique. In: Proceedings of the tenth international conference on information visualisation (IV'06), London, pp 282–290

Risch JS, Rex DB, Dowson ST, Walters TB, May RA, Moon BD (1999) The STARLIGHT information visualization system. In: Card S, Mackinlay J, Shneiderman B (eds) Readings in information visualization: using vision to think. Morgan Kaufmann, San Francisco, pp 551–560

Salton G (1989) Automatic text processing. Addison-Wesley, Reading

Sammon JW (1969) A nonlinear mapping for data structure analysis. IEEE Trans Comput 18(5):401–409

Small D (1996) Navigating large bodies of text. IBM Syst J 35(3&4):514–525

Wise JA, Thomas JJ, Pennock K, Lantrip D, Pottier M, Schur A et al (1995) Visualizing the non-visual: spatial analysis and interaction with information from text documents. In: Proceedings of the IEEE information visualization symposium '95, Atlanta, pp 51–58

York J, Bohn S, Pennock K, Lantrip D (1995) Clustering and dimensionality reduction in SPIRE. In: Proceedings, symposium on advanced information processing and analysis, AIPA95, Tysons Corner

TF–IDF

TF–IDF (*term frequency–inverse document frequency*) is a term weighting scheme commonly used to represent textual documents as vectors (for purposes of classification, clustering, visualization, retrieval, etc.). Let $T = \{t_1, \ldots, t_n\}$ be the set of all terms occurring in the document corpus under consideration. Then a document d_i is represented by a n-dimensional real-valued vector $\mathbf{x}_i = (x_{i_1}, \ldots, x_{in})$ with one component for each possible term from T.

The weight x_{ij} corresponding to term t_j in document d_i is usually a product of three parts: one which depends on the presence or frequency of t_j in d_i, one which depends on t_j's presence in the corpus as a whole, and a normalization part which depends on d_j. The most common TF–IDF weighting is defined by $x_{ij} = \mathrm{TF}_i \cdot \mathrm{IDF}_j \cdot (\sum_j (\mathrm{TF}_{ij} \mathrm{IDF}_j)^2)^{-1/2}$, where TF_{ij} is the *term frequency* (i.e., number of occurrences) of t_j in d_i, and IDFj is the IDF of t_j, defined as $\log(N/\mathrm{DF}_j)$, where N is the number of documents in the corpus and DF_j is the document frequency of t_j (i.e., the number of documents in which t_j occurs). The normalization part ensures that the vector has a Euclidean length of 1.

Several variations on this weighting scheme are also known. Possible alternatives for TF_{ij} include $\min\{1, \mathrm{TF}_{ij}\}$ (to obtain binary vectors) and $(1 + \mathrm{TF}_{ij} / \max_j \mathrm{TF}_{ij})/2$ (to normalize TF within the document). Possible alternatives for IDF_j include 1 (to obtain plain TF vectors instead of TF–IDF vectors) and $\log\,(\sum_i \sum_k \mathrm{TF}_{ik} / \sum_i \mathrm{TF}_{ij})$. The normalization part can be omitted altogether or modified to use some other norm than the Euclidean one.

Threshold Phenomena in Learning

▶ Phase Transitions in Machine Learning

Time Sequence

▶ Time Series

Time Series

Eamonn Keogh
University of California-Riverside, Riverside, CA, USA

Synonyms

Temporal data; Time sequence; Trajectory data

Definition

A *Time Series* is a sequence $T = (t_1, t_2, \ldots, t_n)$ which is an ordered set of n real-valued numbers. The ordering is typically temporal; however, other kinds of data such as color distributions (Hafner et al. 1995), shapes (Ueno et al. 2006), and spectrographs also have a well-defined ordering and can be fruitfully considered "time series" for the purposes of machine learning algorithms.

Motivation and Background

The special structure of time series produces unique challenges for machine learning researchers.

It is often the case that each individual time series object has a very high dimensionality. Whereas classic algorithms often assume a relatively low dimensionality (for example, a few dozen measurements such as "height, weight, blood sugar," etc.), time series learning algorithms must be able to deal with dimensionalities in hundreds or thousands. The problems created by high-dimensional data are more than mere computation time considerations; the very meaning of normally intuitive terms, such as "similar to" and "cluster forming," become unclear in high-dimensional space. The reason for this is that as dimensionality increases, all objects become essentially equidistant to each other and thus classification and clustering lose their meaning. This surprising result is known as the ▶ *curse of dimensionality* and has been the subject of extensive research. The key insight that allows meaningful time series machine learning is that although the actual dimensionality may be high, the *intrinsic* dimensionality is typically much lower. For this reason, virtually all time series data mining algorithms avoid operating on the original "raw" data; instead, they consider some higher level representation or abstraction of the data. Such algorithms are known as ▶ *dimensionality reduction* algorithms. There are many general dimensionality reduction algorithms, such as singular value decomposition and random projections, in addition to many reduction algorithms specifically designed for time series, including piecewise liner approximations, Fourier transforms, wavelets, and symbol approximations (Ding et al. 2008).

In addition to the high dimensionality of individual time series objects, many time series datasets have very high numerosity, resulting in a large volume of data. One implication of high numerosity combined with the high dimensionality of this is that the entire dataset may not fit in main memory. This requires an efficient disk-aware learning algorithm or a careful *sampling* approach.

A final consideration due to the special nature of time series is the fact that individual datapoints are typically highly correlated with their neighbors (a phenomenon known as *autocorrelation*). Indeed, it is this correlation that makes most time series excellent candidates for dimensionality reduction. However, for learning algorithms that assume the independence of features (i.e., ▶ Naive Bayes), this lack of independence must be countered or mitigated in some way.

While virtually every machine learning method has been used to classify time series, the current state-of-the-art method is the nearest neighbor algorithm (Ueno et al. 2006) with a suitable distance measure (Ding et al. 2008). This simple method outperforms neutral networks and Bayesian classifiers.

The major database (SIGMOD, VLDB, PODS) and data mining (SIGKDD, ICDM, SDM) conferences typically feature several time series machine learning/data mining papers each year. In addition, because of the ubiquity of time series, several other communities have active subgroups that conduct research on time series; for example, the SIGGRAPH conference typically has papers on learning or indexing or motion capture time series, and most medical conferences have tracks devoted to medical time series, such as electrocardiograms and electroencephalograms.

The UCR Time Series Archive has several dozen time series datasets which are widely used to test classification and clustering algorithms, and the UCI Data Mining archive has several additional datasets.

Recommended Reading

Ding H, Trajcevski G, Scheuermann P, Wang X, Keogh EA (2008) Querying and mining of time series data: experimental comparison of representations and distance measures. In: Proceeding of the VLDB, Auckland. VLDB endowment

Hafner J, Sawhney H, Equitz W, Flickner M, Niblack W (1995) Efficient color histogram indexing for quadratic form distance functions. IEEE Trans Pattern Anal Mach Intell 17(7):729–736

Ueno K, Xi X, Keogh E, Lee D (2006) Anytime classification using the nearest neighbor algorithm with applications to stream mining. In: Proceedings of IEEE international conference on data mining (ICDM), Hong Kong

Topic Mapping

▶ Text Visualization

Topic Modeling

▶ Topic Models for NLP Applications

Topic Models for NLP Applications

Zhiyuan Chen and Bing Liu
University of Illinois at Chicago, Chicago, IL, USA

Abstract

Topic modeling is a machine learning technique for discovering semantic topics from a document collection. It typically assumes that a document is a multinomial distribution over latent topics, and a topic is a multinomial distribution over words. By capturing the co-occurrence statistics of words in the documents, it uncovers these distributions which indicate important semantic relationships. Topic modeling has been widely studied in machine learning, text mining, and natural language processing (NLP). This chapter gives an introduction to topic modeling. It covers both the fundamental techniques and some of its important applications in NLP.

Synonyms

Topic modeling

Definition

Given a collection of documents, how to discover semantic topics from the documents is an important yet challenging task. It is infeasible to ask human beings to manually read and identify the topics in every available document. This calls for an automated approach to extracting topics. Topic models are statistical machine learning methods that aim to discover a set of latent semantic topics from a document collection or corpus. A topic model is usually represented in a directed graphical model where topics and words are modeled as random variables. In a classic topic model, a document is modeled as an admixture of latent topics, while a topic is regarded as a probability distribution over words. The words are assumed to be generated conditioned on the topics, while topics are assumed to be sampled from a predefined distribution. Topic models are based on “higher-order co-occurrence,” i.e., how often words co-occur in different contexts. They usually perform well with a large number of documents which provide reliable co-occurrence statistics.

Motivation and Background

Discovering semantic topics from text corpora is beneficial to many applications in natural language processing. Due to the wide variety, high volume, and dynamic nature of topics, manual topic identification is clearly not scalable. To address it, topic models, such as latent Dirichlet allocation (LDA) (Blei et al. 2003) and probabilistic latent semantic analysis (pLSA) (Hofmann 1999), have been proposed to automatically discover latent topics from text corpora. In general, topic models assume that each document is a multinomial distribution over topics, while each semantic topic is a multinomial distribution over words. The two types of resulting distributions in topic models are document-topic distributions and topic-word distributions, respectively. The intuition is that certain words are more or less likely to be present given the topics of a document. For example, “sport” and “player” will appear more often in documents about sports, and “rain” and “cloud” will appear more frequently in documents about weather.

Structure of the Learning System

Topic modeling represents a class of statistical methods that can automatically extract thematic information from unstructured text documents. Topic models usually assume a generative process to describe how words are generated in documents. We use the most popular topic model, LDA (latent Dirichlet allocation) (Blei et al. 2003), as an example to explain. We denote the number of documents by M and the number of topics by T. Each document $m \in \{1, \ldots, M\}$

contains N_m words. The vocabulary in the corpus is denoted by $\{1, \ldots, V\}$. The generative process of LDA is given as follows:

1. For each topic $t \in \{1, \ldots, T\}$
   (i) Draw a per topic distribution over words, $\varphi_t \sim Dir(\beta)$
2. For each document $m \in \{1, \ldots, M\}$
   (i) Draw a topic distribution, $\theta_m \sim Dir(\alpha)$
   (ii) For each word position n in document m, where $n \in \{1, \ldots, N_m\}$
      (a) Draw a topic $z_{m,n} \sim Mult(\theta_m)$
      (b) Emit word $w_{m,n} \sim Mult(\varphi_{z_{m,n}})$

Here, α and β are called Dirichlet priors representing hyperparameters. $Dir()$ denotes the Dirichlet distribution and $Mult()$ indicates the multinomial distribution. Note that Dirichlet distribution is the conjugate prior of the multinomial distribution which simplifies the model inference derivation. θ is the document-topic distribution and φ is the topic-word distribution.

Inference and Parameter Estimation

The posterior inference of the LDA model is intractable and cannot be solved by exact inference. Common approximate inference techniques include collapsed Gibbs sampling (Griffiths and Steyvers 2004), variational methods (Blei et al. 2003), and expectation propagation (Minka and Lafferty 2002). Collapsed Gibbs sampling is the most popular inference approach due to its simplicity.

Gibbs sampling is a special case of Metropolis-Hastings algorithm which is a MCMC (Markov chain Monte Carlo) technique. It is usually used to generate samples from a joint probability of many random variables to approximate the marginal distribution. Gibbs sampling is especially useful when it is hard to sample from the joint distribution directly due to its complexity, but sampling from the conditional distribution of the random variables is easy. It is an iterative process that starts with a random initialization of the Markov chain's state. In each iteration, the value of each random variable is updated by drawing a sample from its conditional distribution based on the current state of all other random variables and the data.

The conditional distribution of assigning topic t to a word w_i in the collapsed Gibbs sampler for LDA is stated as below:

$$P(z_i = t | z^{-i}, \boldsymbol{w}, \alpha, \beta) \propto \frac{n_{m,t}^{-i} + \alpha}{\sum_{t'=1}^{T} (n_{m,t'}^{-i} + \alpha)} \times \frac{n_{t,w_i}^{-i} + \beta}{\sum_{v'=1}^{V} (n_{t,v'}^{-i} + \beta)} \tag{1}$$

where z^{-i} are the topic assignments excluding the current topic assignment of w_i. $\boldsymbol{w}$ denotes all the words in the documents. n^{-i} is the count that excludes the current word. $n_{m,t}$ is the number of times that topic t appears in document m, and $n_{t,w}$ is the number of occurrences of word w under topic t. Equation 1 is quite intuitive: the first ratio expresses the probability of topic t in document m, and the second ratio implies the probability of word w under topic t. Since this information is sufficient to compute the conditional distribution, Gibbs sampling can be implemented efficiently by caching and updating these counts only.

The estimation of document-topic distribution θ and topic-word distribution φ is straightforward given the samples of Gibbs sampling as below:

$$\theta_{m,t} = \frac{n_{m,t} + \alpha}{\sum_{t'=1}^{T} (n_{m,t'} + \alpha)} \tag{2}$$

$$\varphi_{t,w} = \frac{n_{t,w} + \beta}{\sum_{v'=1}^{V} (n_{t,v'} + \beta)} \tag{3}$$

Nonparametric Topic Models

Classic topic models such as LDA and pLSA generally require the number of topics to be specified by the user before running the actual models. In practice, this number is usually set empirically by conducting some initial experiments, which may not guarantee the optimal parameters for the model. Nonparametric topic

T

models automatically learn the appropriate number of topics from the data itself without manual setting for the number of topics. The hierarchical Dirichlet process mixture model (Teh et al. 2006) is the first nonparametric topic model. It introduces the Dirichlet process into topic models to automatically estimate the number of topics. The intuition is that there is a set of infinite groups in the data where each observation is generated independently given a group. This is the same as LDA as there is a set of topics where each word is sampled given a topic except that LDA fixes the number of topics, while the hierarchical Dirichlet process mixture model assumes that there are infinite topics. For the inference of the hierarchical Dirichlet process mixture model, Gibbs sampling is also used. More details can be found in Teh et al. (2006).

Knowledge-Based Topic Models

Most of the traditional topic models are fully unsupervised. However, researchers have shown that fully unsupervised topic models often produce incoherent topics because the objective functions of topic models do not always correlate well with human judgments (Chang et al. 2009). To address this issue, several knowledge-based topic models (KBTM), also called semi-supervised topic models, have been proposed and used in NLP applications.

DF-LDA (Andrzejewski et al. 2009) is the first KBTM which incorporates prior knowledge in the forms of must-links and cannot-links where a must-link states that two words should belong to the same topic and a cannot-link states that two words should not be in the same topic. In a similar but more generic vein, must-sets and cannot-sets are used in MC-LDA (Chen et al. 2013). Mukherjee and Liu (2012) proposed a model that allows the human user to provide some seed words in some topics. Interactive topic models were also proposed to allow the user to interact with the model during its inference process (Hu et al. 2011). In Blei and McAuliffe (2010), document class labels were considered in a supervised setting. However, these works in KBTM require the user to be involved to provide the knowledge or guidance for a superior model performance. As we know, expert knowledge can be hard to obtain. To address it, lifelong topic modeling (LTM) (Chen and Liu 2014) was proposed to automatically mine the prior knowledge from past domains and leverage the knowledge to help discover topics of higher quality in a new domain.

Applications in NLP

Since topic models are primarily designed for analysis of text documents, there are numerous applications in almost every subarea of NLP. It is difficult to describe them all. Here we discuss only a few subareas to give a flavor of the types of NLP applications.

Part-of-speech (POS) tagging is one of core NLP tasks. The task is to specify a particular part of speech to a given word based on the definition and context of that word. For example, in a sentence “Bob enjoys reading books,” the word *enjoys* is marked up with POS tag *VBZ*, indicating that this word is a verb with third-person singular present. The challenge is that some words may have multiple POS tags, e.g., the word *move* can be a verb or a noun. In such cases, the context of the given word is usually required to decide the correct POS tag.

Topic models have been widely applied in the task of POS tagging. Griffiths et al. (2004) proposed a topic model that can model both the semantic and syntactic information for part-of-speech tagging. Their motivation is that a word in a sentence can have one of the two roles: serving a syntactic function or providing a semantic meaning (Griffiths et al. 2004). Syntactic words usually have short-range dependencies, i.e., spanning several words without going beyond the scope of a sentence. In contrast, semantic words tend to have long-range dependencies: some sentences within a document are likely to share similar words and express similar contexts. Based on it, a hidden Markov model (HMM) was used inside the generative model to decide whether a word belongs to the syntactic class or the semantic class. Obtaining such information is helpful in determining POS tags. For example, knowing that a word "control" in a text corpus belongs to the syntactic class makes it more likely to be a verb than a noun. Toutanova and Johnson (2008) further added a sparse prior to topic models on the distribution over tags for each word. They also explicitly modeled ambiguity classes, i.e., the set of part-of-speech tags that a word can be associated with. In their model, each word type is assigned with a set of possible parts of speech, and each token of this word type is associated with a part-of-speech tag.

Word sense disambiguation (WSD) is another important NLP area where topic models have been popularly applied. Its objective is to identify the sense or meaning of an ambiguous word in its context. For example, the word *light* in "the light of the sun" refers to the meaning "something that makes things visible," while *light* in "The box is light to carry" indicates the sense "of little weight." A dictionary, e.g., WordNet (https://wordnet.princeton.edu/), is usually used to help provide word senses. Boyd-Graber et al. (2007) proposed a model called LDAWN (LDA with WordNet) model to distinguish word senses. In WordNet, a word sense is represented by a synset (short for synonym sets). For example, in the above examples, the synset {light, luminance} is associated with the sense of "something that makes things visible." LDAWN models the synset path, i.e., a path from one synset to another synset, as a hidden variable. It assumes that words under the same topic are likely to share the same meaning as well as their synset path. The posterior inference of LDAWN was conducted using Gibbs sampling to infer the synset path, i.e., the sense, of a word. The key advantage of LDAWN is that it does not need labeled data to disambiguate a corpus. It simultaneously decomposes a corpus into topics with words grouping into their word senses.

Sentiment analysis (or opinion mining) is perhaps one of the biggest application areas of topic models in NLP. The goal of sentiment analysis is to extract subjective information such as opinions, evaluations, appraisals, and emotions from text. Liu (2012) gave a comprehensive survey of the sentiment analysis and opinion mining research. Topic models have been widely applied in aspect-based sentiment analysis, which is a fine-grained analysis of opinions, to infer aspects and opinion words. Aspects in the sentiment analysis context are entity features on which opinions have been expressed. For example, in a review sentence, "The picture looks great," about a camera, the aspect is "picture" and the opinion word is "great." Mei et al. (2007) proposed the topic-sentiment mixture (TSM) model to reveal the latent topics and their associated sentiments in a Weblog collection. They also designed a special HMM structure in the topic model to detect topic life cycles and sentiment dynamics. A semi-supervised topic model was proposed in Lu and Zhai (2008) to integrate opinions expressed in well-written expert reviews and opinions expressed by the general public in sources such as weblogs to generate an aligned and integrated opinion summary. Titov and McDonald (2008) proposed a topic model to distinguish global aspects and local aspects. In their model, global aspects correspond to global properties of objects, e.g., the brand of a product type, while local aspects are the aspects of an object or entity that tend to be rated or evaluated by users.

More recently, Lin and He (2009) proposed the joint sentiment/topic (JST) model that jointly models topics (aspects) and sentiments. Rather than having only one set of latent topic variables as in LDA, JST adds another set of hidden sentiment variables. The advantage of JST is that it is able to model both aspects and sentiments in a fully unsupervised fashion without the need of supervised information such as labels. Based on JST, Jo and Oh (2011) made an assumption that one sentence represents only one aspect, i.e., all the words in a sentence are generated from one aspect. However, these models do not actually separate aspects and opinion words in their results. The maximum entropy model was integrated into a topic model by Zhao et al. (2010) to explicitly separate opinions from aspects. Chen et al. (2014) proposed the AKL (automated knowledge LDA) model that learns prior knowledge from reviews of other products/domains and applies such knowledge to mine more coherent aspects. The knowledge base is represented by a set of clusters, where each cluster consists of words that are semantically correlated.

Some other NLP applications of topic models include machine translation (Eidelman et al. 2012), summarization (Haghighi and Vanderwende 2009), tagging (Krestel et al. 2009), multi-language topic synchronization (Petterson et al. 2010), topical keyphrase extraction (Zhao et al. 2011), relation extraction between named entities (Yao et al. 2011), entity linking (Han and Sun 2012), and document retrieval (Wei and Croft 2006).

Cross-References

- ▶ Bayesian Network
- ▶ Graphical Models
- ▶ Unsupervised Learning

Recommended Reading

Andrzejewski D, Zhu X, Craven M (2009) Incorporating domain knowledge into topic modeling via Dirichlet Forest priors. In: ICML, Montreal, pp 25–32

Blei DM, McAuliffe JD (2010) Supervised topic models. In: NIPS, Whistler, pp 121–128

Blei DM, Ng AY, Jordan MI (2003) Latent Dirichlet allocation. J Mach Learn Res 3:993–1022

Boyd-Graber JL, Blei DM, Zhu X (2007) A topic model for word sense disambiguation. In: EMNLP-CoNLL, Prague, pp 1024–1033

Chang J, Boyd-Graber J, Chong W, Gerrish S, Blei DM (2009) Reading tea leaves: how humans interpret topic models. In: NIPS, Whistler, pp 288–296

Chen Z, Liu B (2014) Topic modeling using topics from many domains, lifelong learning and big data. In: ICML, Beijing, pp 703–711

Chen Z, Mukherjee A, Liu B, Hsu M, Castellanos M, Ghosh R (2013) Exploiting domain knowledge in aspect extraction. In: EMNLP, Seattle, pp 1655–1667

Chen Z, Mukherjee A, Liu B (2014) Aspect extraction with automated prior knowledge learning. In: ACL, Baltimore, pp 347–358

Eidelman V, Boyd-Graber J, Resnik P (2012) Topic models for dynamic translation model adaptation. In: ACL, Jeju Island, pp 115–119

Griffiths TL, Steyvers M (2004) Finding scientific topics. PNAS 101(Suppl):5228–5235

Griffiths TL, Steyvers M, Blei DM, Tenenbaum JB (2004) Integrating topics and syntax. In: NIPS, Vancouver, pp 537–544

Haghighi A, Vanderwende L (2009) Exploring content models for multi-document summarization. In: ACL, Boulder, pp 362–370

Han X, Sun L (2012) An entity-topic model for entity linking. In: EMNLP, Jeju Island, pp 105–115

Hofmann T (1999) Probabilistic latent semantic analysis. In: UAI, Stockholm, pp 289–296

Hu Y, Boyd-Graber J, Satinoff B (2011) Interactive topic modeling. In: ACL, Portland, pp 248–257

Jo Y, Oh AH (2011) Aspect and sentiment unification model for online review analysis. In: WSDM, Hong Kong, pp 815–824

Krestel R, Fankhauser P, Nejdl W (2009) Latent dirichlet allocation for tag recommendation. In: RecSys, New York, pp 61–68

Lin C, He Y (2009) Joint sentiment/topic model for sentiment analysis. In: CIKM, Hong Kong, pp 375–384

Liu B (2012) Sentiment analysis and opinion mining. Synth Lect Hum Lang Technol 5(1):1–167

Lu Y, Zhai C (2008) Opinion integration through semi-supervised topic modeling. In: WWW, Beijing, pp 121–130

Mei Q, Ling X, Wondra M, Su H, Zhai C (2007) Topic sentiment mixture: modeling facets and opinions in weblogs. In: WWW, Banff, pp 171–180

Minka T, Lafferty J (2002) Expectation-propagation for the generative aspect model. In: UAI'02, Edmonton, pp 352–359

Mukherjee A, Liu B (2012) Aspect extraction through semi-supervised modeling. In: ACL, Jeju Island, pp 339–348

Petterson J, Smola A, Caetano T, Buntine W, Narayanamurthy S (2010) Word features for latent Dirichlet allocation. In: NIPS, Whistler, pp 1921–1929

Teh YW, Jordan MI, Beal MJ, Blei DM (2006) Hierarchical Dirichlet processes. J Am Stat Assoc 101(476): 1–30

Titov I, McDonald R (2008) Modeling online reviews with multi-grain topic models. In: WWW, Beijing, pp 111–120

Toutanova K, Johnson M (2008) A Bayesian LDA-based Model for Semi-Supervised Part-of-speech Tagging. In: NIPS, Whistler

Wei X, Croft WB (2006) LDA-based document models for ad-hoc retrieval. In: SIGIR, Seattle, pp 178–185

Yao L, Haghighi A, Riedel S, McCallum A (2011) Structured relation discovery using generative models. In: EMNLP, Edinburgh, pp 1456–1466

Zhao WX, Jiang J, He J, Song Y, Achananuparp P, Lim E-P, Li X (2011) Topical keyphrase extraction from twitter. In: ACL, Portland, pp 379–388

Zhao WX, Jiang J, Yan H, Li X (2010) Jointly modeling aspects and opinions with a MaxEnt-LDA hybrid. In: EMNLP, Cambridge, pp 56–65

Topology

▶ Topology of a Neural Network

Topology of a Neural Network

Risto Miikkulainen
Department of Computer Science, The
University of Texas at Austin, Austin, TX, USA

Synonyms

Architecture; Connectivity; Structure; Topology

Definition

Topology of a neural network refers to the way the neurons are connected, and it is an important factor in how the network functions and learns. A common topology in unsupervised learning is a direct mapping of inputs to a collection of units that represents categories (e.g., ▶ Self-Organizing Maps). The most common topology in supervised learning is the fully connected, three-layer, feedforward network (see ▶ Backpropagation and ▶ Radial Basis Function Networks): All input values to the network are connected to all neurons in the hidden layer (hidden because they are not visible in the input or output), the outputs of the hidden neurons are connected to all neurons in the output layer, and the activations of the output neurons constitute the output of the whole network. Such networks are popular partly because they are known theoretically to be universal function approximators (with, e.g., a sigmoid or Gaussian nonlinearity in the hidden layer neurons), although networks with more layers may be easier to train in practice (e.g., ▶ Cascade-Correlation). In particular, deep learning architectures (see ▶ Deep Learning) utilize multiple hidden layers to form a hierarchy of gradually more structured representations that support a supervised task on top. Layered networks can be extended to processing sequential input and/or output by saving a copy of the hidden layer activations and using it as additional input to the hidden layer in the next time step (see ▶ Simple Recurrent Network). Fully recurrent topologies, where each neuron is connected to all other neurons (and possibly to itself), can also be used to model time-varying behavior, although such networks may be unstable and difficult to train (e.g., with backpropagation; but see also ▶ Boltzmann Machines). Modular topologies, where different parts of the networks perform distinctly different tasks, can improve stability and can also be used to model high-level behavior (e.g., ▶ Echo-State Machines and ▶ Adaptive Resonance Theory). Whatever the topology, in most cases, learning involves modifying the ▶ Weight on the network connections. However, arbitrary network topologies are possible as well and can be constructed as part of the learning (e.g., with backpropagation or ▶ Neuroevolution) to enhance feature selection, recurrent memory, abstraction, or generalization.

Trace-Based Programming

Pierre Flener[1] and Ute Schmid[2]
[1]Department of Information Technology,
Uppsala University, Uppsala, Sweden
[2]Faculty of Information Systems and Applied
Computer Science, University of Bamberg,
Bamberg, Germany

Abstract

Trace-based programming is introduced as a specific approach to inductive programming where a, typically recursive, program is inferred from a small set of example computational traces.

Most of the work by this author was done while on leave of absence in 2006/2007 as a Visiting Faculty Member and Erasmus Exchange Teacher at Sabancı University, Turkey.

Synonyms

Programming from traces

Definition

Trace-based programming addresses the inference of a program from a small set of example computation traces. The induced program is typically a recursive program. A computation *trace* is a non-recursive expression that describes the transformation of some specific input into the desired output with help of a predefined set of primitive functions. While the construction of traces is highly dependent on background knowledge or even on knowledge about the program searched for, the inductive **generalization** is based on syntactical methods of detecting regularities and dependencies between traces, as proposed in classical approaches to ▶ inductive programming or ▶ explanation-based learning. As an alternative to providing traces by hand simulation, AI planning techniques or **programming by demonstration** can be used.

Cross-References

▶ Explanation-Based Learning
▶ Inductive Programming
▶ Programming by Demonstration

Recommended Reading

Biermann AW (1972) On the inference of Turing machines from sample computations. Artif Intell 3(3):181–198

Schmid U, Wysotzki F (1998) Induction of recursive program schemes. In: Proceedings of the 10th European conference on machine learning (ECML 1998). Volume 1398 of lecture notes in artificial intelligence. Springer, pp 214–225

Schrödl S, Edelkamp S (1999) Inferring flow of control in program synthesis by example. In: Proceedings of the 23rd annual German conference on artificial intelligence (KI 1999). Volume 1701 of lecture notes in artificial intelligence. Springer, pp 171–182

Shavlik JW (1990) Acquiring recursive and iterative concepts with explanation-based learning. Mach Learn 5:39–70

Wysotzki F (1983) Representation and induction of infinite concepts and recursive action sequences. In: Proceedings of the 8th international joint conference on artificial intelligence (IJCAI 1983). Morgan Kaufmann, pp 409–414

Training Curve

▶ Learning Curves in Machine Learning

Training Data

Synonyms

Training examples; Training instances

Definition

Training data are data to which a learner is applied.

Cross-References

▶ Training Set

Training Examples

▶ Training Data

Training Instances

▶ Training Data

Training Set

Synonyms

Training data

Definition

A training set is a ▶ data set containing data that are used for learning by a learning system. A training set may be divided further into a ▶ growing set and a ▶ pruning set.

Cross-References

▶ Data Set
▶ Training Data

Training Time

A learning algorithm is typically applied at two distinct times. Training time refers to the time when an algorithm is learning a model from ▶ training data. ▶ Test time refers to the time when an algorithm is applying a learned model to make predictions. ▶ Lazy learning usually blurs the distinction between these two times, deferring most learning until test time.

Trait

▶ Attribute

Trajectory Data

▶ Time Series

Transductive Learning

▶ Semi-supervised Learning
▶ Semi-supervised Text Processing

Transfer Learning

▶ Inductive Transfer

Transfer of Knowledge Across Domains

▶ Inductive Transfer

Transition Probabilities

In a ▶ Markov decision process, the *transition probabilities* represent the probability of being in *state* s' at time $t+1$, given you take *action* a from state s at time t for all s, a and t.

Tree Augmented Naive Bayes

Fei Zheng[1,2] and Geoffrey I. Webb[3]
[1]Monash University, Sydney, NSW, Australia
[2]Monash University, Clayton, Melbourne, VIC, Australia
[3]Faculty of Information Technology, Monash University, Victoria, Australia

Synonyms

TAN

Definition

Tree augmented ▶ naive Bayes is a ▶ semi-naive Bayesian Learning method. It relaxes the naive Bayes attribute independence assumption by employing a tree structure, in which each attribute only depends on the class and one other attribute. A maximum weighted spanning tree that maximizes the likelihood of the training data is used to perform classification.

Classification with TAN

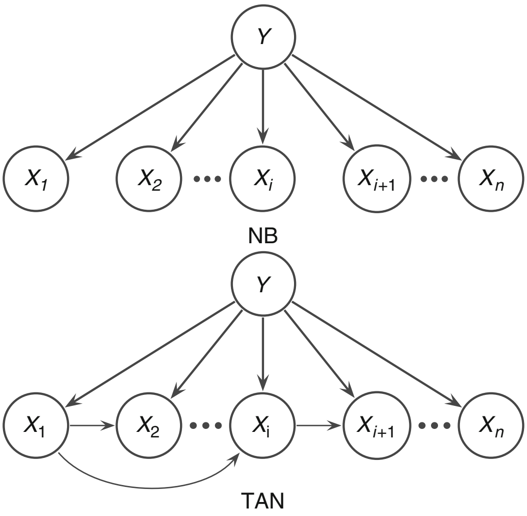
Interdependencies between attributes can be addressed directly by allowing an attribute to depend on other non-class attributes. However, techniques for learning unrestricted Bayesian networks often fail to deliver lower zero-one loss than naive Bayes (Friedman et al. 1997). One possible reason for this is that full ▶ Bayesian networks are oriented toward optimizing the likelihood of the training data rather than the conditional likelihood of the class attribute given a full set of other attributes. Another possible reason is that full Bayesian networks have high variance due to the large number of parameters estimated. An intermediate alternative technique is to use a less restrict structure than naive Bayes. Tree augmented naive Bayes (TAN) (Friedman et al. 1997) employs a tree structure, allowing each attribute to depend on the class and at most one other attribute. Figure 1 shows Bayesian network representations of the types of model that NB and TAN respectively create.

Chow (1968) proposed a method that efficiently constructs a maximum weighted spanning tree which maximizes the likelihood that the training data was generated from the tree. The weight of an edge in the tree is the mutual information of the two attributes connected by the edge. TAN extends this method by using conditional mutual information as weights. Since the selection of root does not affect the log-likelihood of the tree, TAN randomly selects a root attribute and directs all edges away from it. The parent of each attribute X_i is indicated as $\pi(X_i)$ and the parent of the class is $\varnothing$. It assumes that attributes are independent given the class and their parents and classifies the test instance $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ by selecting

$$\underset{y}{\operatorname{argmax}}\ \hat{P}(Y) \prod_{1 \le i \le n} \hat{P}(x_i | y, \pi(x_i)), \qquad (1)$$

where $\pi(x_i)$ is a value of $\pi(X_i)$ and y is a class label.

Due to the relaxed attribute independence assumption, TAN considerably reduces the ▶ bias of naive Bayes at the cost of an increase in variance. Empirical results (Friedman et al. 1997) show that it substantially reduces zero-one loss of naive Bayes on many data sets and that of all data sets examined it achieves lower zero-one loss than naive Bayes more often than not.



Tree Augmented Naive Bayes, Fig. 1 Bayesian network examples of the forms of model created by NB and TAN

Cross-References

- ▶ Averaged One-Dependence Estimators
- ▶ Bayesian Network
- ▶ Naïve Bayes
- ▶ Semi-Naive Bayesian Learning

Recommended Reading

Chow CK, Liu CN (1968) Approximating discrete probability distributions with dependence trees. IEEE Trans Inf Theory 14:462–467

Friedman N, Geiger D, Goldszmidt M (1997) Bayesian network classifiers. Mach Learn 29(2):131–163

Tree Mining

Siegfried Nijssen
Katholieke Universiteit Leuven, Leuven,
Belgium

Definition

Tree mining is an instance of constraint-based pattern mining and studiesthe discovery of tree patterns in data that is represented as a tree structure or as a set of trees structures. Minimum frequency is the most studied constraint.

Motivation and Background

Tree mining is motivated by the availability of many types of data that can be represented as tree structures. There is a large variety in tree types, for instance, ordered trees, unordered trees, rooted trees, unrooted (free) trees, labeled trees, unlabeled trees, and binary trees; each of these has its own application areas. An example are trees in tree banks, which store sentences annotated with parse trees. In such data, it is not only of interest to find commonly occurring sets of words (for which frequent itemset miners could be used), but also to find commonly occurring parses of these words. Tree miners aim at finding patterns in this structured information. The patterns can be interesting in their own right, or can be used as features in classification algorithms.

Structure of Problem

All tree miners share a similar problem setting. Their input consists of a set of trees and a set of constraints, usually a minimum frequency constraint, and their output consists of all subtrees that fulfill the constraints.

Tree miners differ in the constraints that they are able to deal with, and the types of trees that they operate on. The following types of trees can be distinguished:

Free trees, which are graphs without cycles, and no order on the nodes or edges;

Unordered trees, which are free trees in which one node is chosen to be the root of the tree;

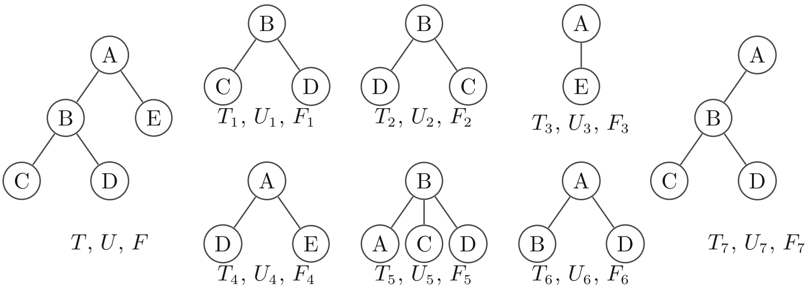
Ordered trees, which are rooted trees in which the nodes are totally ordered.

For each of these types of tree, we can choose to have labels on the nodes, or on the edges, or on both.

The differences between these types of trees are illustrated in Fig. 1. Every graph in this figure can be interpreted as a free tree F_i, an unordered tree U_i, or an ordered tree T_i. When interpreted as ordered trees, none of the trees are equivalent. When we interpret them as unordered trees, U_1 and U_2 are equivalent representations of the same unordered tree that has B as its root and C and D as its children. Finally, as free trees, not only F_1 and F_2 are equivalent, but also F_5 and F_7.

Intuitively, a free tree requires less specification than an ordered tree. The number of possible free trees is smaller than the number of possible ordered trees. On the other hand, to test if two trees are equivalent we need a more elaborate computation for free trees than for ordered trees.

Assume that we have data represented as (a set of) trees, then the data mining problem is to find patterns, represented as trees, that fulfill constraints based on this data. To express these constraints, we need a coverage relation that expresses when one tree can be considered to occur in another tree. Different coverage relations can be expressed for free trees, ordered trees, and unordered trees. We will introduce these relations through operations that can be used to transform



Tree Mining, Fig. 1 The leftmost tree is part of the data, the other trees could be patterns in this tree, depending on the subtree relation that is used

Tree Mining, Fig. 2 Relations between the trees of Fig. 1

Tree	*Induced*	*Embedded*	*Root-Induced*	*Bottom-up*	*Prefix*	*Leaf-set*
T_1	yes	yes	no	yes	no	no
T_2	no	no	no	no	no	no
T_3	yes	yes	yes	no	no	yes
T_4	no	yes	no	no	no	no
T_5	no	no	no	no	no	no
T_6	no	no	no	no	no	no
T_7	yes	yes	yes	no	yes	yes

Tree	*Induced*	*Embedded*	*Root-Induced*	*Bottom-up*	*Leaf-set*
U_1	yes	yes	no	yes	no
U_2	yes	yes	no	yes	no
U_3	yes	yes	yes	no	yes
U_4	no	yes	no	no	no
U_5	no	no	no	no	no
U_6	no	no	no	no	no
U_7	yes	yes	yes	no	yes

Tree	*Induced*
F_1	yes
F_2	yes
F_3	yes
F_4	no
F_5	yes
F_6	no
F_7	yes

trees. As an example, consider the operation that removes a leaf from a tree. We can repeatedly apply this operation to turn a large tree into a smaller one. Given two trees A and B, we say that A occurs in B as

Induced subtree, if A can be obtained from B by repeatedly removing leaves from B. When dealing with rooted trees, the root is here also considered to be a leaf if it has one child;

Root-induced subtree, if A can be obtained from B by repeatedly removing leaves from B. When dealing with rooted trees, the root is not allowed to be removed;

Embedded subtree, if A can be obtained from B by repeatedly either (1) removing a leaf or (2) removing an internal node, reconnecting the children of the removed node with the parent of the removed node;

Bottom-up subtree, if there is a node v in B such that if we remove all nodes from B that are not a descendant of v, we obtain A;

Prefix, if A can be obtained from B by repeatedly removing the last node from the ordered tree B;

Leaf set, if A can be obtained from B by selecting a set of leaves from B, and all their ancestors in B.

For free trees, only the induced subtree relation is well-defined. A prefix is only well-defined for ordered trees, the other relations apply both to ordered and unordered trees. In the case of unordered trees, we assume that each operation maintains the order of the original tree B. The relations are also illustrated in Fig. 2.

Intuitively, we can speak of *occurrences* (also called *embeddings* by some authors) of a small tree in a larger tree. Each such occurrence (or embedding) can be thought of as a function φ that maps every node in the small tree to a node in the large tree.

Using an occurrence relation, we can define frequency measures. Assume given a forest $\mathcal{F}$ of trees, all ordered, unordered, or free. Then the frequency of a tree A can be defined

Transaction-based, where we count the number of trees $B \in \mathcal{F}$ such that A is a subtree of B;

Node-based, where we count the number of nodes v in $\mathcal{F}$ such that A is a subtree of the bottom-up subtree below v.

Node-based frequency is only applicable in rooted trees, in combination with the root-induced, bottom-up, prefix, or leaf set subtree relations.

Given a definition of frequency, constraints on trees of interest can be expressed:

Minimum frequency, to specify that only trees with a certain minimum number of occurrences are of interest;

Closedness, to specify that a tree is only of interest if its frequency is different from all its supertrees;

Maximality, to specify that a tree is only of interest if none of its supertrees is frequent.

Observe that in all of these constraints, the subtree relation is again important. The subtree relation is not only used to compare patterns with data, but also patterns among themselves.

The tree mining problem can now be stated as follows. Given a forest of trees $\mathcal{F}$ (ordered, unordered, or free) and a set of constraints, based on a subtree relation, the task is to find all trees that satisfy the given constraints.

Theory/Solution

The tree mining problem is an instance of the more general problem of constraint-based pattern mining under constraints. For more information about the general setting, see the sections on constraint-based mining, itemset mining, and graph mining.

All algorithms iterate a process of generating candidate patterns, and testing if these candidates satisfy the constraints. Essential is to avoid that every possible tree is considered as a candidate. To this purpose, the algorithms exploit that many frequency measures are anti-monotonic. This property states that for two given trees A and B, where A is a subtree of B, if A is infrequent, then also B is infrequent, and therefore, we do not need to consider it as a candidate.

This observation can make it possible to find all trees that satisfy the constraints, if these requirements are fulfilled:

- We have an algorithm to enumerate candidate subtrees, which satisfies these properties:
  - It should be able to enumerate all trees in the search space;
  - It should avoid that no two equivalent subtrees are listed;
  - It should only list a tree after at least one of its subtrees has been listed, to exploit the anti-monotonicity of the frequency constraint;
- We have an algorithm to efficiently compute in how many database trees a pattern tree occurs.

The algorithmic solutions to these problems depend on the type of tree and the subtree relation.

Encoding and Enumerating Trees

We will first consider how tree miners internally represent trees. Two types of encodings have been proposed, both of which are string-based. We will illustrate these encodings for node-labeled trees, and start with *ordered* trees.

The first encoding is based on a *preorder* listing of nodes: (1) for a rooted ordered tree T with a single vertex r, the *preorder string* of T is $S_{T,r} = l_r - 1$, where l_r is the label for the single vertex r, and (2) for a rooted ordered tree T with more than one vertex, assuming the root of T is r (with label l_r) and the children of r are $r_1, \ldots, r_K$ from left to right, then the preorder string for T is

$S_{T,r} = l_r S_{T,r_K} - 1$, where $S_T, r_1, \dots, S_T, r_k$ are the preorder strings for the bottom-up subtrees below nodes $r_1, \dots, r_K$ in T.

The second encoding is based on listing the *depths* of the nodes together with their labels in prefix-order. The depth of a node v is the length of the path from the root to the node v. The code for a tree is $S_{T,r} = d_r, l_r, S_{T,r_1} \dots S_{T,r_k}$, where d_r is the depth of the node r in tree T.

Both encodings are illustrated in Fig. 3.

A search space of trees can be visualized as in Fig. 4. In this figure, every node corresponds to the depth encoding of a tree, while the edges visualize the partial order defined by the subtree relation. It can be seen that the number of induced subtree relations between trees is smaller than the number of embedded subtree relations.

Tree	Depth-sequence	Preorder string
T_6	$1A2B2D$	AB-1D-1
T_7	$1A2B3C3D$	ABC-1D-1-1-1
T	$1A2B3C3D2E$	ABC-1D-1-1E-1
T_4	$1A2D2E$	AD-1E-1-1
T_3	$1A2E$	AE-1-1
T_5	$1B2A2C2D$	BA-1C-1D-1-1
T_1	$1B2C2D$	BC-1D-1-1
T_2	$1B2D2C$	BD-1C-1-1

Tree Mining, Fig. 3 Depth sequences for all the trees of Fig. 1, sorted in lexicographical order. Tree T_2 is the canonical form of unordered tree U_2, as its depth sequence is the highest among equivalent representations

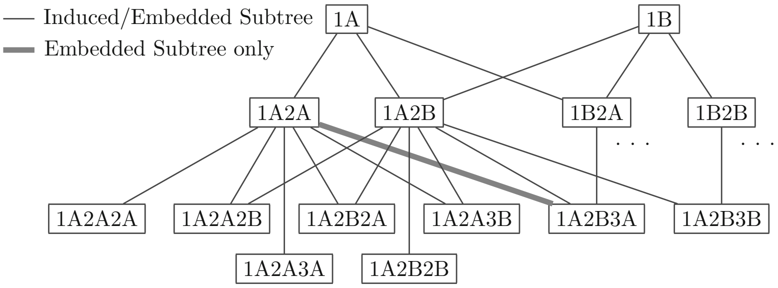
The task of the enumeration algorithm is to traverse this search space starting from trees that contain only one node. Most algorithms perform the search by building an enumeration tree over the search space. In this enumeration tree every pattern should have a single parent. The children of a pattern in the enumeration tree are called its *extensions* or its *refinements*. An example of an enumeration tree for the induced subtree relation is given in Fig. 5.

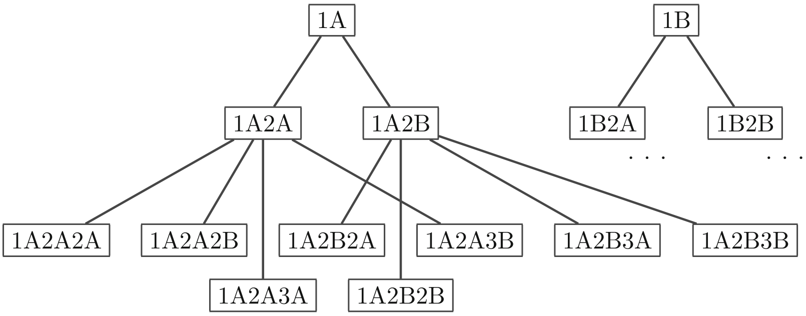
In the enumeration tree that is given here, the parent of a tree is its prefix in the depth encoding. An alternative definition is that the parent of a tree can be obtained by removing the last node in a prefix order traversal of the ordered tree. Every refinement in the enumeration has one additional node that is connected to the *rightmost path* of the parent.

The enumeration problem is more complicated for *unordered trees*. In this case, the trees represented by the strings $1A2A2B$ and $1A2B2A$ are equivalent, and we only wish to enumerate one of these strings. This can be achieved by defining a total order on all strings that represent trees, and to define that only the highest (or lowest) string of a set of equivalent strings should be considered.

For depth encodings, the ordering is usually lexicographical, and the highest string is chosen to be the *canonical* encoding. In our example, $1A2B2A$ would be canonical. This code has the desirable property that every prefix of a canonical code is also a canonical code. Furthermore it can be determined in polynomial time which

Tree Mining, Fig. 4 A search space of ordered trees, where edges denote subtree relationships





Tree Mining, Fig. 5 Part of an enumeration tree for the search space of Fig. 4

extensions of a canonical code lead to a canonical code, such that it is not necessary to consider any code that is not canonical.

Alternative codes have also been proposed, which are not based on a preorder, depth-first traversal of a tree, but on a level-wise listing of the nodes in a tree.

Finally, for *free trees* we have the additional problem that we de not have a root for the tree. Fortunately, it is known that every free tree either has a uniquely determined *center* or a uniquely determined *bicenter*. This (bi)center can be found by determining the longest path between two nodes in a free tree: the node(s) in the middle of this path are the center of the tree. It can be shown that if multiple paths have the same maximal length, they will have the same (bi)center. By appointing one center to be the root, we obtain a rooted tree, for which we can compute a code.

To avoid that two codes are listed that represent equivalent free trees, several solutions have been proposed. One is based on the idea of first enumerating paths (thus fixing the center of a tree), and for each of these paths enumerating all trees that can be grown around them. Another solution is based on enumerating all rooted, unordered trees under the constraint that at least two different children of the root have a bottom-up subtree of equal, maximal depth. In the first approach, a preorder depth encoding was used; in the second approach a level-wise encoding was used.

Counting Trees

To evaluate the frequency of a tree the subtree relation between a candidate pattern tree and all trees in the database has to be computed. For each of our subtree relations, polynomial algorithms are known to decide the relation, which are summarized in Table 1.

Tree Mining, Table 1 Worst case complexities of the best known algorithms that determine whether a tree relation holds between two trees; m is the number of nodes in the pattern tree, l is the number of leafs in the pattern tree, n the number of nodes in the database tree

Ordered	
Embedding	$O(nl)$
Induced	$O(nm)$
Root-induced	$O(n)$
Leaf-set	$O(n)$
Bottom-up	$O(n)$
Prefix	$O(m)$
Unordered	
Embedding	NP-complete
Induced	$O(nm^{1\frac{1}{2}}/\log m)$
Root-induced	$O(nm^{1\frac{1}{2}}/\log m)$
Leaf-set	$O(nm^{1\frac{1}{2}}/\log m)$
Bottom-up	$O(n)$

Even though a subtree testing algorithm and an algorithm for enumerating trees are sufficient to compute all frequent subtrees correctly, in practice fine-tuning is needed to obtain an efficient method. There are two reasons for this:

- In some databases, the number of candidates can by far exceed the number of trees that are actually frequent. One way to reduce the number of candidates is to only generate a particular candidate after we have encountered at least one occurrence of it in the data (this is called *pattern growth*); another way is to require that a candidate is only generated if at least two of its subtrees satisfy the constraints (this is called *pattern joining*).

- The trees in the search space are very similar to each other: a parent only differs from its children by the absence of a single node. If memory allows, it is desirable to *reuse* the subtree matching information, instead of starting the matching from scratch.

A large number of data structures have been proposed to exploit these observations. We will illustrate these ideas using the FreqT algorithm, which mines induced, ordered subtrees, and uses a depth encoding for the trees.

In FreqT, for a given pattern tree A, a list of (database tree, database node) pointers is stored. Every element (B, v) in this list corresponds to an occurrence of tree A in tree B in which the last node (in terms of the preorder) of A is mapped to node v in database tree B. For a database and three example trees this is illustrated in Fig. 6.

Every tree in the database is stored as follows. Every node is given an index, and for every node, we store the index of its parent, its righthand sibling, and its first child.

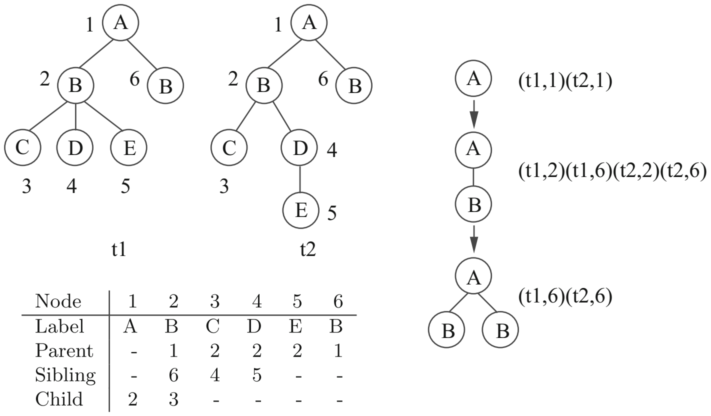
Let us consider how we can compute the occurrences of the subtree $1A2B2B$ from the occurrences of the tree $1A2B$. The first occurrence of $1A2B$ is $(t1, 2)$, which means that the B labeled node can be mapped to node 2 in $t1$. Using the arrays that store the database tree, we can then conclude that node 6, which is the right-hand sibling of node 2, corresponds to an occurence of the subtree $1A2B2B$. Therefore, we add $(t1, 6)$ to the occurrence list of $1A2B2B$. Similarly, by scanning the data we find out that the first child of node 2 corresponds to an occurrence of the subree $1A2B3C$, and we add $(t1, 3)$ to the occurrence list of $1A2B3C$.

Overall, using the parent, sibling and child pointers we can scan every node in the data that could correspond to a valid expansion of the subtree $1A2B$, and update the corresponding lists. After we have done this for every occurrence of the subtree, we know the occurrence lists of all possible extensions.

From an occurrence list we can determine the frequency of a tree. For instance, the transaction-based frequency can be computed by counting the number of different database trees occurring in the list.

As we claimed, this example illustrates two features that are commonly seen in tree miners: first, the occurrence list of one tree is used to compute the occurrence list of another tree, thus reusing information; second, the candidates are collected from the data by scanning the nodes that connect to the occurrence of a tree in the data. Furthermore, this example illustrates that a careful design of the datastructure that stores the data can ease the frequency evaluation considerably.

FreqT does not perform pattern joining. The most well-known example of an algorithm that



Node	1	2	3	4	5	6
Label	A	B	C	D	E	B
Parent	-	1	2	2	2	1
Sibling	-	6	4	5	-	-
Child	2	3	-	-	-	-

Tree Mining, Fig. 6 A tree database (*left*) and three ordered trees with their occurrence lists according to the FreqT algorithm (*right*). The datastructure that stores $t1$ in FreqT is given in the table (*right*) performs tree joining is the embedded TreeMiner (Zaki 2002). Both the FreqT and the TreeMiner perform the search depth-first, but also tree miners that use the traditional level-wise approach of the Apriori algorithm have been proposed. The FreqT and the TreeMiner have been extended to unordered trees.

Other Constraints

As the number of frequent subtrees can be very large, approaches have been studied to reduce the number of trees returned by the algorithm, of which closed and maximal trees are the most popular. To find closed or maximal trees, two issues need to be addressed:

- How do we make sure that we only output a tree if it is closed or maximal, that is, how do we determine that none of its supertrees has the same support, or is frequent?
- Can we conclude that some parts of the search space will never contain a closed or maximal tree, thus making the search more efficient?

Two approaches can be used to address the first issue:

- All closed patterns can be stored, and every new pattern can be compared with the stored set of patterns;
- When we evaluate the frequency of a pattern in the data, we also (re)evaluate the frequency of all its possible extensions, and only output the pattern if its support is different.

The second approach requires less memory, but in some cases requires more computations.

To prune the search space, a common approach is to check all occurrences of a tree in the data. If every occurrence of a tree can be extended into an occurrence of another tree, the small tree should not be considered, and the search should continue with the tree that contains all common edges and nodes. Contrary to graph mining, it can be shown that this kind of pruning can safely be done in most cases.

Applications

Examples of databases to which tree mining algorithms have been applied are

Parse tree analysis: Since the early 1990s large *Treebank* datasets have been collected consisting of sentences and their grammatical structure. An example is the Penn TreeBank (Marcus et al. 1993). These databases contain rooted, ordered trees. To discover differences in domain languages it is useful to compare commonly occurring grammatical constructions in two different sets of parsed texts, for which tree miners can be used (Sekine 1998).

Computer network analysis: IP *multicast* is a protocol for sending data to multiple receivers. In an IP multicast session a webserver sends a packet once; routers copy a packet if two different routes are required to reach multiple receivers. During a multicast session rooted trees are obtained in which the root is the sender and the leaves are the receivers. Commonly occurring patterns in the routing data can be discovered by analyzing these unordered rooted trees (Chalmers and Almeroth 2003).

Webserver access log analysis: When users browse a website, this behavior is reflected in the access log files of the webserver. Servers collect information such as the webpage that was visited, the time of the visit, and the webpage that was clicked to reach the webpage. The access logs can be transformed into a set of ordered trees, each of which corresponds to a visitor. Nodes in these trees correspond to webpages; edges are inserted if a user browses from one webpage to another. Nodes are ordered in viewing order. A tool was developed to perform this transformation in a sensible way (Punin et al. 2002).

Phylogenetic trees: One of the largest tree databases currently under construction is the TreeBASE database, which is comprised of a large number of phylogenetic trees (Morell 1996). The trees in the TreeBASE database are submitted by researchers and are collected from publications. Originating from multiple sources, they can disagree on parts of the phylogenetic tree. To find common agreements between the trees, tree miners have been used (Zhang and Wang 2005). The phylogenetic trees are typically unordered; labels among siblings are unique.

Hypergraph mining: Hypergraphs are graphs in which one edge can have more than two endpoints. Those hypergraphs in which no two nodes share the same label can be transformed into unordered trees, as follows. First, an artificial root is inserted. Second, for each edge of the hypergraph a child node is added to the root, labeled with the label of the hyperedge. Finally, the labels of nodes within hyperedges are added as leaves to the tree. An example of hypergraph data is bibliographic data: if each example corresponds to a paper, nodes in the hypergraph correspond to authors cited by the paper, and hyperedges connect coauthors of cited papers.

Multi-relational data mining: Many multi-relational databases are tree shaped, or a tree-shaped view can be created. For instance, a transaction database in which every transaction is associated with customers and their information, can be represented as a tree (Berka 1999).

XML data mining: Several authors have stressed that tree mining algorithms are most suitable for mining XML data. XML is a tree–shaped data format, and tree miners can be helpful when trying to (re)construct Document Type Definitions (DTDs) for such documents.

Cross-References

▶ Constraint-Based Mining
▶ Graph Mining

Further Reading

The FreqT algorithm was introduced in (Asai et al. 2002; Wang and Liu 1998; Zaki 2002). The most popular tree miner is the embedded tree miner by Zaki (2002). A more detailed overview of tree miners can be found in Chi et al. (2005). Most implementations of tree miners are available on request from their authors.

Recommended Reading

Asai T, Abe K, Kawasoe S, Arimura H, Satamoto H, Arikawa S (2002) Efficient substructure discovery from large semi-structured data. In: Proceedings of the second SIAM international conference on data mining, Arlington. SIAM, pp 158–174

Berka P (1999) Workshop notes on discovery challenge PKDD-99 (Technical report). University of Economics, Prague

Chalmers R, Almeroth K (2003) On the topology of multicast trees. IEEE/ACM Trans Netw 11:153–165. IEEE Press/ACM Press

Chi Y, Nijssen S, Muntz RR, Kok JN (2005) Frequent subtree mining—an overview. In: Fundam Inform 66:161–198. IOS Press

Marcus MP, Santorini B, Marcinkiewicz MA (1993) Building a large annotated corpus of English: the Penn Treebank. Comput Linguist 19:313–330. MIT Press

Morell V (1996) TreeBASE: the roots of phylogeny. Science 273:569

Punin J, Krishnamoorthy M, Zaki MJ (2002) LOGML—log markup language for web usage mining. In: WEBKDD 2001—mining web log data across all customers touch points. Third international workshop, San Francisco. Lecture notes in artificial intelligence, vol 2356. Springer, pp 88–112

Sekine S (1998) Corpus-based parsing and sublanguages studies. Ph.D. dissertation. New York University, New York

Wang K, Liu H (1998) Discovering typical structures of documents: a road map approach. In: Proceedings of the 21st annual international ACM SIGIR conference on research and development in information retrieval, Melbourne. ACM Press, pp 146–154

Zaki MJ (2002) Efficiently mining frequent trees in a forest. In: Proceedings of the 8th international conference knowledge discovery and data mining (KDD), Edmonton. ACM Press, pp 71–80

Zhang S, Wang J (2005) Frequent agreement subtree mining. http://aria.njit.edu/mediadb/fast/

Tree-Based Regression

▶ Regression Trees

True Lift Modeling

► Uplift Modeling

True Negative

True negatives are the negative examples that are correctly classified by a classification model. See ► confusion matrix for a complete range of related terms.

True Negative Rate

► Specificity

True Positive

True positives are the positive examples that are correctly classified by a classification model. See ► confusion matrix for a complete range of related terms.

True Positive Rate

► Sensitivity

Type

► Class

Typical Complexity of Learning

► Phase Transitions in Machine Learning

U

Underlying Objective

The term *objective* used in Evolutionary Multi-Objective Optimization refers to an indicator of quality returning an element from an ordered set of scalar values, such as a real number. For any test-based coevolutionary problem, a set of underlying objectives exists such that knowledge of the objective values of an individual is sufficient to determine the outcomes of all possible tests. The existence of a set of underlying objectives is guaranteed, as the set of possible tests itself satisfies this property.

Unit

▸ Neuron

Universal Learning Theory

Marcus Hutter
Research School of Computer Science,
Australian National University, Canberra, ACT,
Australia

Abstract

This encyclopedic article gives a mini-introduction into the theory of universal learning, founded by Ray Solomonoff in the 1960s and significantly developed and extended in the last decade. It explains the spirit of universal learning, but necessarily glosses over technical subtleties.

Definition, Motivation, and Background

Universal (machine) learning is concerned with the development and study of algorithms that are able to learn from data in a very large range of environments with as few assumptions as possible. The class of environments typically considered includes all computable stochastic processes. The investigated learning tasks range from ▸ inductive inference, sequence prediction, sequential decisions, to (re)active problems such as ▸ reinforcement learning (Hutter 2005), but also include ▸ clustering, ▸ regression, and others (Li and Vitányi 2008). Despite various ▸ no free lunch theorems, universal learning is *possible* by assuming that the data possess *some* effective structure, but without specifying any further *which* structure (Lattimore and Hutter 2011). Learning algorithms that are universal (at least to some degree) are also *necessary* for developing autonomous general intelligent systems, required, e.g., for exploring other planets, as opposed to decision *support* systems which keep a human in the loop. There is also an *intrinsic* interest in striving for generality: Finding new learning algorithms for every particular (new) problem is possible but cumbersome and prone

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

to disagreement or contradiction. A sound formal general and ideally complete theory of learning can unify existing approaches, guide the development of practical learning algorithms, and last but not least lead to novel and deep insights.

Deterministic Environments

Let $t, n \in \mathbb{N}$ be natural numbers, $\mathcal{X}^*$ be the set of finite strings, and $\mathcal{X}^\infty$ be the set of infinite sequences over some alphabet $\mathcal{X}$ of size $|\mathcal{X}|$. For a string $x \in \mathcal{X}^*$ of length $\ell(x) = n$, we write $x_1x_2\ldots x_n$ with $x_t \in \mathcal{X}$ and further abbreviate $x_{t:n} := x_t x_{t+1}\ldots x_{n-1}x_n$ and $x_{<n} := x_1\ldots x_{n-1}$, and $\epsilon = x_{<1}$ for the empty string. Consider a countable class of deterministic hypotheses $\mathcal{M} = \{H_1, H_2, \ldots\}$. Each hypothesis $H \in \mathcal{M}$ (also called model) shall describe an infinite sequence $x^H_{1:\infty}$, e.g., like in IQ test questions "2,4,6,8,…." In online learning, for $t = 1, 2, 3, \ldots$, we predict x_t based on past observations $\dot{x}_{<t}$, then nature reveals $\dot{x}_t$, and so on, where the dot above x indicates the true observation. We assume that the true hypothesis is in $\mathcal{M}$, i.e., $\dot{x}_{1:\infty} = x^{H_m}_{1:\infty}$ for some $m \in \mathbb{N}$. The goal is to ("quickly") identify the unknown H_m from the observations.

Learning by enumeration works as follows: Let $\mathcal{M}_t = \{H \in \mathcal{M} : x^H_{<t} = \dot{x}_{<t}\}$ be the set of hypotheses consistent with our observations $\dot{x}_{<t}$ so far. The hypothesis in $\mathcal{M}_t$ with smallest index, say m'_t, is selected and used for predicting x_t. Then $\dot{x}_t$ is observed and all $H \in \mathcal{M}_t$ inconsistent with x_t are eliminated, i.e., they are not included in $\mathcal{M}_{t+1}$. Every prediction error results in the elimination of at least $H_{m'_t}$, so after at most $m-1$ errors, the true hypothesis H_m gets selected forever, since it never makes an error ($H_m \in \mathcal{M}_t \forall t$). This identification may take arbitrarily long (in t), but the number of errors on the way is bounded by $m-1$, and the latter is often more important. As an example for which the bound is attained, consider H_i with $x^{H_i}_{1:\infty} := 1^{f(i)}0^\infty$ $\forall i$ for any strictly increasing function f, e.g., $f(i) = i$. But we now show that we can do much better than this, at least for finite $\mathcal{X}$.

Majority learning. Consider (temporarily in this paragraph only) a binary alphabet $\mathcal{X} = \{0, 1\}$ and a *finite* deterministic hypothesis class $\mathcal{M} = \{H_1, H_2, \ldots, H_N\}$. H_m and $\mathcal{M}_t$ are as before, but now we take a majority vote among the hypotheses in $\mathcal{M}_t$ as our prediction of x_t. If the prediction turns out to be wrong, then at least half (the majority) of the hypotheses get eliminated from $\mathcal{M}_t$. Hence after at most $\log N$ errors, there is only a single hypothesis, namely, H_m, left over. So this majority predictor makes at most $\log N$ errors. As an example where this bound is essentially attained, consider $m = N = 2^n - 1$ and let $x^{H_i}_{1:\infty}$ be the digits after the comma of the binary expansion of $(i-1)/2^n$ for $i = 1, \ldots, N$.

Weighted majority for countable classes. Majority learning can be adapted to denumerable classes $\mathcal{M}$ and general finite alphabet $\mathcal{X}$ as follows: Each hypothesis H_i is assigned a weight $w_i > 0$ with $\sum_i w_i \leq 1$. Let $W := \sum_{i:H_i \in \mathcal{M}_t} w_i$ be the total weight of the hypotheses in $\mathcal{M}_t$. Let $\mathcal{M}^a_t := \{H_i \in \mathcal{M}_t : x^{H_i}_t = a\}$ be the consistent hypotheses predicting $x_t = a$, and W_a their weight, and take the weighted majority prediction $x_t = \arg\max_a W_a$. Similarly as above, a prediction error decreases W by a factor of $1 - 1/|\mathcal{X}|$, since $\max_a W_a \geq W/|\mathcal{X}|$. Since $w_m \leq W \leq 1$, this algorithm can at most make $\log_{1-1/|\mathcal{X}|} w_m = O(\log w_m^{-1})$ prediction errors. If we choose, for instance, $w_i = (i+1)^{-2}$, the number of errors is $O(\log m)$, which is an exponential improvement over the Gold-style learning by enumeration above.

Algorithmic Probability

Algorithmic probability has been founded by Ray Solomonoff (1964). The so-called universal probability or a priori probability is the key quantity for universal learning. Its philosophical and technical roots are ▶ Ockham's razor (choose the simplest model consistent with the data), Epicurus' principle of multiple explanations (keep all explanations consistent with the data), (Universal) Turing machines (to compute, quantify, and assign codes to all quantities of interest), and Kolmogorov complexity (to define what simplicity/complexity means) (Rathmanner and Hutter 2011). This section considers deterministic computable sequences and the next section the general setup of computable probability distributions.

(Universal) monotone Turing machines. Since we consider infinite computable sequences, we need devices that convert input data streams to output data streams. For this we define the following variants of a classical deterministic Turing machine: A monotone Turing machine T is defined as a Turing machine with one unidirectional input tape, one unidirectional output tape, and some bidirectional work tapes (Li and Vitányi 2008). The input tape is binary (no blank) and read only, the output tape is over finite alphabet $\mathcal{X}$ (no blank) and write only, unidirectional tapes are those where the head can only move from left to right, work tapes are initially filled with zeros, and the output tape with some fixed element from $\mathcal{X}$. We *say* that *monotone Turing machine* T outputs/computes a string starting with $x \in \mathcal{X}^*$ on input $p \in \{0,1\}^*$ and write $T(p) = x*$ if p is to the left of the input head when the last symbol of x is output (T reads all of p but no more). T may continue operation and need not halt. For a given x, the set of such p forms a prefix code. Such codes are called *minimal* programs. Similarly we write $T(p) = \omega$ if p outputs the infinite sequence ω. A *prefix code* $\mathcal{P}$ is a set of binary strings such that no element is proper prefix of another. It satisfies *Kraft's inequality* $\sum_{p\in\mathcal{P}} 2^{-\ell(p)} \leq 1$.

The table of rules of a Turing machine T can be prefix encoded in a canonical way as a binary string, denoted by $\langle T \rangle$. Hence, the set of Turing machines $\{T_1, T_2, \ldots\}$ can be effectively enumerated. There are so-called universal Turing machines that can "simulate" all other Turing machines. We define a particular one which simulates monotone Turing machine $T(q)$ if fed with input $\langle T \rangle q$, i.e., $U(\langle T \rangle q) = T(q)\ \forall T, q$. Note that for p not of the form $\langle T \rangle q$, $U(p)$ does not output anything. We call this particular U the *reference universal Turing machine*.

Universal weighted majority learning. $T_1(\epsilon), T_2(\epsilon), \ldots$ constitutes an effective enumeration of all finite and infinite computable sequences, hence so does monotone $U(p)$ for $p \in \{0,1\}^*$. As argued below, the class of computable infinite sequences is conceptually very interesting. The halting problem implies that there is no recursive enumeration of all partial recursive functions with infinite domain; hence we cannot remove the finite sequences algorithmically. It is very fortunate that we don't have to. Hypothesis H_p is identified with the sequence $U(p)$, which may be finite, infinite, or possibly even empty. The class of considered hypotheses is $\mathcal{M} := \{H_p : p \in \{0,1\}^*\}$.

The weighted majority algorithm also needs weights w_p for each H_p. Ockham's razor combined with Epicurus' principle demand to assign a high (low) prior weight to a simple (complex) hypothesis. If complexity is identified with program length, then w_p should be a decreasing function of $\ell(p)$. It turns out that $w_p = 2^{-\ell(p)}$ is the "right" choice, since minimal p form a prefix code and therefore $\sum_p w_p \leq 1$ as required.

Using H_p for prediction can now fail in two ways. H_p may make a wrong prediction or no prediction at all for x_t. The true hypothesis H_m is still assumed to produce an infinite sequence. The weighted majority algorithm in this setting makes at most $O(\log w_p^{-1}) = O(\ell(p))$ errors. It is also plausible that learning $\ell(p)$ bits requires $O(\ell(p))$ "trials."

Universal mixture prediction. Solomonoff (1978) defined the following universal a priori probability:

$$M(x) := \sum_{p:U(p)=x*} 2^{-\ell(p)} \tag{1}$$

That is, $M(x) = W$ is the total weight of the computable deterministic hypotheses consistent with x for the universal weight choice $w_p = 2^{-\ell(p)}$. The universal weighted majority algorithm predicted $\arg\max_a M(\dot{x}_{<t}a)$. Instead, one could also make a probability prediction $M(a|\dot{x}_{<t}) := M(\dot{x}_{<t}a)/M(\dot{x}_{<t})$, which is the relative weight of hypotheses in $\mathcal{M}_t$ predicting a. The higher the probability $M(\dot{x}_t|\dot{x}_{<t})$ that is assigned to the true next observation $\dot{x}_t$, the better. Consider the absolute prediction error $|1 - M(\dot{x}_t|\dot{x}_{<t})|$ and the logarithmic error$-\log M(\dot{x}_t|\dot{x}_{<t})$. The cumulative logarithmic error is bounded by $\sum_{t=1}^{n} -\log M(\dot{x}_t|\dot{x}_{<t}) = -\log M(\dot{x}_{1:n}) \leq \ell(p)$ for any program p that prints $\dot{x}_{1:n}*$. For instance, p could be chosen as the shortest one printing $\dot{x}_{1:\infty}$, which has length $Km(\dot{x}_{1:\infty}) := \min\{\ell(p) : U(p) = \dot{x}_{1:\infty}\}$. Using $1-z \leq -\log z$ and letting $n \to \infty$, we get

$$\sum_{t=1}^{\infty}|1 - M(\dot{x}_t|\dot{x}_{<t})| \leq \sum_{t=1}^{\infty} -\log M(\dot{x}_t|\dot{x}_{<t}) \leq Km(\dot{x}_{1:\infty}) \qquad (2)$$

Hence again, the cumulative absolute and logarithmic errors are bounded by the number of bits required to describe the true environment.

Universal Bayes

The exposition so far has dealt with deterministic environments only. Data sequences produced by real-world processes are rarely as clean as IQ test sequences. They are often noisy. This section deals with stochastic sequences sampled from computable probability distributions. The developed theory can be regarded as an instantiation of Bayesian learning. Bayes' theorem allows to update beliefs in face of new information but is mute about how to choose the prior and the model class to begin with. Subjective choices based on prior knowledge are informal, and traditional "objective" choices such as Jeffreys prior are not universal (Rathmanner and Hutter 2011). Machine learning, the computer science branch of statistics, develops (fully) automatic inference and decision algorithms for very large problems. Naturally, machine learning has (re)discovered and exploited different principles (Ockham's and Epicurus') for choosing priors, appropriate for this situation. This leads to an alternative representation of universal probability as a mixture over all lower semi-computable semimeasures with Kolmogorov complexity based prior as described below.

Bayes. Sequences $\omega = \omega_{1:\infty} \in \mathcal{X}^\infty$ are now assumed to be sampled from the "true" probability measure μ, i.e., $\mu(x_{1:n}) := \mathrm{P}[\omega_{1:n} = x_{1:n}|\mu]$ is the μ-probability that ω starts with $x_{1:n}$. Expectations w.r.t. μ are denoted by $\mathbf{E}$. In particular for a function $f : \mathcal{X}^n \to \mathbb{R}$, we have $\mathbf{E}[f] = \mathbf{E}[f(\omega_{1:n})] = \sum_{x_{1:n}} \mu(x_{1:n})f(x_{1:n})$. Note that in Bayesian learning, measures, environments, and models are the same objects; let $\mathcal{M} = \{\nu_1, \nu_2, \ldots\} \equiv \{H_{\nu_1}, H_{\nu_2}, \ldots\}$ denote a countable class of these measures≡hypotheses. Assume that μ is unknown but known to be a member of $\mathcal{M}$, and $w_\nu := \mathrm{P}[H_\nu]$ is the given prior belief in H_ν. Then the Bayes mixture

$$\xi(x_{1:n}) := \mathrm{P}[\omega_{1:n} = x_{1:n}] = \sum_{\nu\in\mathcal{M}} \mathrm{P}[\omega_{1:n} = x_{1:n}|H_\nu]\mathrm{P}[H_\nu] \equiv \sum_{\nu\in\mathcal{M}} \nu(x_{1:n})w_\nu \qquad (3)$$

must be our a priori belief in $x_{1:n}$, and $\mathrm{P}[H_\nu|\omega_{1:n} = x_{1:n}] = w_\nu\nu(x_{1:n})/\xi(x_{1:n})$ be our posterior belief in ν by Bayes' rule.

Universal choice of $\mathcal{M}$. Next we need to find a universal class of environments $\mathcal{M}_U$. Roughly speaking, Bayes works if $\mathcal{M}$ contains the true environment μ. The larger the $\mathcal{M}$, the less restrictive is this assumption. The class of all computable distributions, although only countable, is pretty large from a practical point of view, since it includes for instance all of today's valid physics theories. (Finding a non-computable physical system would indeed overturn the generally accepted Church-Turing thesis.) It is the largest class, relevant from a computational point of view. Solomonoff (1964, Eq. (13)) defined and studied the mixture over this class.

One problem is that this class is not (effectively = recursively) enumerable, since the class of computable functions is not enumerable due to the halting problem, nor is it decidable whether a function is a measure. Hence ξ is completely incomputable. Leonid Levin (Li and Vitányi 2008) had the idea to "slightly" extend the class and include also lower semi-computable semimeasures.

A function $\nu : \mathcal{X}^* \to [0,1]$ is called a semimeasure iff $\nu(x) \geq \sum_{a\in\mathcal{X}} \nu(xa) \forall x \in \mathcal{X}^*$. It is a proper probability measure iff equality holds and $\nu(\epsilon) = 1$. $\nu(x)$ still denotes the ν-probability that a sequence starts with string x. A function is called lower semi-computable if it can be approximated from below. Similarly to that fact that the class of partial recursive functions is recursively enumerable, one can show that the class $\mathcal{M}_U = \{\nu_1, \nu_2, \ldots\}$ of lower semi-computable semimeasures is recursively enumerable. In some sense $\mathcal{M}_U$ is the largest class of environments for which ξ is in some sense computable, but even larger classes are possible (Schmidhuber 2002).

Kolmogorov complexity. Before we can turn to the prior w_ν, we need to quantify complexity/simplicity. Intuitively, a string is simple if it can be described in a few words, such as "the string of one million ones," and is complex if there is no such short description, like for a random object whose shortest description is specifying it bit by bit. We are interested in effective descriptions and hence restrict decoders to be Turing machines. One can define the *prefix Kolmogorov complexity* of string x as the length ℓ of the shortest *halting* program p for which U outputs x:

$$K(x) := \min_p\{\ell(p) : U(p) = x \text{ halts}\}$$

Simple strings such as 000…0 can be generated by short programs and, hence, have low Kolmogorov complexity, but irregular (e.g., random) strings are their own shortest description and hence have high Kolmogorov complexity. For non-string objects o (such as numbers and functions), one defines $K(o) := K(\langle o\rangle)$, where $\langle o\rangle \in \mathcal{X}^*$ is some standard code for o. In particular, $K(\nu_i) = K(i)$.

To be brief, K is an excellent universal complexity measure, suitable for quantifying Ockham's razor.

The universal prior. We can now quantify a prior biased toward simple models. First, we quantify the complexity of an environment ν or hypothesis H_ν by its Kolmogorov complexity $K(\nu)$. The universal prior should be a decreasing function in the model's complexity and of course sum to (less than) one. Since $\sum_x 2^{-K(x)} \leq 1$ by the prefix property and Kraft's inequality, this suggests the choice

$$w_\nu = w_\nu^U := 2^{-K(\nu)} \qquad (4)$$

Since $\log i \leq K(\nu_i) \leq \log i + 2\log\log i$ for "most" i, most ν_i have prior approximately reciprocal to their index i as also advocated by Jeffreys and Rissanen.

Representations. Combining the universal class $\mathcal{M}_U$ with the universal prior (4), we arrive at the universal mixture

$$\xi_U(x) := \sum_{\nu\in\mathcal{M}_U} 2^{-K(\nu)}\nu(x) \qquad (5)$$

which has remarkable properties. First, it is itself a lower semi-computable semimeasure, that is, $\xi_U \in \mathcal{M}_U$, which is very convenient. Note that for most classes, $\xi \notin \mathcal{M}$.

Second, ξ_U coincides with M (Wood et al. 2011) and $M \in \mathcal{M}_U$. This means that the mixture over deterministic computable sequences is as rich as the mixture over the much larger class of semi-computable semimeasures. The intuitive reason is that the probabilistic semimeasures are in the convex hull of the deterministic ones and therefore need not be taken extra into account in the mixture.

There is another, possibly the simplest, representation: One can show that $M(x)$ is equal to the probability that U outputs a string starting with x when provided with uniform random noise on the program tape. Note that a uniform distribution is also used in many no free lunch theorems to prove the impossibility of universal learners, but in our case, the uniform distribution is piped through a universal Turing machine, which defeats these negative implications as we will see in the next section (Lattimore and Hutter 2011).

Applications

In the stochastic case, identification of the true hypothesis is problematic. The posterior P$[H|x]$ may not concentrate around the true hypothesis H_μ if there are other hypotheses H_ν that are not asymptotically distinguishable from H_μ. But even if model identification (*induction* in the narrow sense) fails, *predictions*, *decisions*, and *actions* can be good, and indeed, for universal learning, this is generally the case.

Universal sequence prediction. Given a sequence $x_1x_2\ldots x_{t-1}$, we want to predict its likely continuation x_t. We assume that the strings which have to be continued are drawn from a computable "true" probability distribution μ. The maximal prior information a prediction algorithm can possess is the exact knowledge of μ, but often the true distribution is unknown. Instead, prediction is based on a guess ρ of μ. Let $\rho(a|x) := \rho(xa)/\rho(x)$ be the *predictive* ρ-probability that the next symbol is $a \in \mathcal{X}$, given sequence $x \in \mathcal{X}^*$. Since $\mu \in \mathcal{M}_U$ it is natural to use ξ_U or M for prediction.

Solomonoff's (1978, 2005) celebrated result indeed shows that M converges to μ. For general alphabet it reads

$$\sum_{t=1}^{\infty}\mathbf{E}\Big[\sum_{a\in\mathcal{X}}\big(M(a|\omega_{<t})-\mu(a|\omega_{<t})\big)^2\Big] \leq K(\mu)\,\text{In}\,2+O(1) \qquad (6)$$

Analogous bounds hold for ξ_U and for other than the Euclidian distance, e.g., the Hellinger and the absolute distance and the relative entropy.

For a sequence $a_1, a_2, \ldots$ of random variables, $\sum_{t=1}^{\infty}\mathbf{E}[a_t^2] \leq c < \infty$ implies $a_t \to 0$ for $t \to \infty$ with μ-probability 1 (w.p.1). Convergence is rapid in the sense that the probability that a_t^2 exceeds $\varepsilon > 0$ at more than $c/\varepsilon\delta$ times is bounded by δ. This might loosely be called the number of errors. Hence Solomonoff's bound implies

$$M(x_t|\omega_{<t}) - \mu(x_t|\omega_{<t}) \longrightarrow 0 \quad \text{for any } x_t \text{ rapid w.p.1 for } t\to\infty \qquad (7)$$

The number of times M deviates from μ by more than $\varepsilon > 0$ is bounded by $O(K(\mu))$, i.e., is proportional to the complexity of the environment, which is again reasonable. A counting argument shows that $O(K(\mu))$ errors for most μ are unavoidable. No other choice for w_ν would lead to significantly better bounds. Again, in general it is not possible to determine *when* these "errors" occur. Multistep lookahead convergence $M(x_{t:n_t}|\omega_{<t}) - \mu(x_{t:n_t}|\omega_{<t}) \to 0$ even for unbounded lookahead $n_t - t \geq 0$, relevant for delayed sequence prediction and in reactive environments, can also be shown.

In summary, M is an excellent sequence predictor under the only assumption that the observed sequence is drawn from some (unknown) computable probability distribution. No ergodicity, stationarity, or identifiability or other assumption is required.

Universal sequential decisions. Predictions usually form the basis for decisions and actions, which result in some profit or loss. Let $\ell_{x_t y_t} \in [0,1]$ be the received loss for decision $y_t \in \mathcal{Y}$ when $x_t \in \mathcal{X}$ turns out to be the true tth symbol of the sequence. The ρ-optimal strategy

$$y_t^{\Lambda_\rho}(\omega_{<t}) := \arg\min_{y_t}\sum_{x_t}\rho(x_t|\omega_{<t})\ell_{x_t y_t} \qquad (8)$$

minimizes the ρ-expected loss. For instance, if we can decide among $\mathcal{Y} = \{\textit{sunglasses}, \textit{umbrella}\}$

and it turns out to be $\mathcal{X} = \{sun, rain\}$, and our personal loss matrix is $\ell = \left(\begin{smallmatrix} 0.0 & 0.1 \\ 1.0 & 0.3 \end{smallmatrix}\right)$, then Λ_ρ takes $y_t^{\Lambda_\rho}$ =*sunglasses* if $\rho(rain|\omega_{<t}) < {}^1\!/_8$ and an *umbrella* otherwise. For $\mathcal{X} = \mathcal{Y}$ and 0–1 loss $\ell_{xy} = 0$ for $x = y$ and 1 else, Λ_ρ predicts the most likely symbol $y_t^{\Lambda_\rho} = \arg\max_a \rho(a|\omega_{<t})$ as in section "Deterministic Environments".

The cumulative μ(= true)-expected loss of Λ_ρ for the first n symbols is

$$\text{Loss}_n^{\Lambda_\rho} := \sum_{t=1}^{n} \mathbf{E}[\ell_{\omega_t y_t^{\Lambda_\rho}(\omega_{<t})}]$$

$$\equiv \sum_{t=1}^{n} \sum_{x_{1:t}} \mu(x_{1:t}) \ell_{x_t y_t^{\Lambda_\rho}(x_{<t})} \qquad (9)$$

If μ is known, Λ_μ obviously results in the best decisions in the sense of achieving minimal expected loss among all strategies. For the predictor Λ_M based on M (and similarly ξ_U), one can show (Hutter 2007)

$$\sqrt{\text{Loss}_n^{\Lambda_M}} - \sqrt{\text{Loss}_n^{\Lambda_\mu}} \le \sqrt{2K(\mu)\ \text{In}\ 2 + O(1)} \qquad (10)$$

This implies that $\text{Loss}_n^{\Lambda_M}/\text{Loss}_n^{\Lambda_\mu} \to 1$ for $\text{Loss}_n^{\Lambda_\mu} \to \infty$, or if $\text{Loss}_\infty^{\Lambda_\mu}$ is finite, then also $\text{Loss}_\infty^{\Lambda_M} < \infty$. This shows that M (via Λ_M) performs also excellent from a decision-theoretic perspective, i.e., suffers loss only slightly larger than the optimal Λ_μ strategy.

One can also show that Λ_M is Pareto optimal (admissible) in the sense that every other predictor with smaller loss than Λ_M in some environment $\nu \in \mathcal{M}_U$ must be worse in another environment.

Universal classification and regression. The goal of classification and regression is to infer the functional relationship $f : \mathcal{Y} \to \mathcal{X}$ from data $\{(y_1, x_1), \ldots, (y_n, x_n)\}$. In a predictive online setting, one wants to "directly" infer x_t from y_t given $(y_{<t}, x_{<t})$ for $t = 1, 2, 3, \ldots$. The universal induction framework has to be extended by regarding $y_{1:\infty}$ as independent side information presented in the form of an oracle or extra tape information or extra parameter. The construction has to ensure that $x_{1:n}$ only depends on $y_{1:n}$ but is (functionally or statistically) independent of $y_{n+1:\infty}$.

First, we augment a monotone Turing machine with an extra input tape containing $y_{1:\infty}$. The Turing machine is called chronological if it does not read beyond $y_{1:n}$ before $x_{1:n}$ has been written. Second, semimeasures $\rho = \mu, \nu, M, \xi_U$ are extended to $\rho(x_{1:n}|y_{1:\infty})$, i.e., one semimeasure $\rho(\cdot|y_{1:\infty})$ for each $y_{1:\infty}$ (no distribution over y is assumed, despite the suggestive |). Any such semimeasure must be chronological in the sense that $\rho(x_{1:n}|y_{1:\infty})$ is independent of y_t for $t>n$; hence we can write $\rho(x_{1:n}|y_{1:n})$. In classification and regression, ρ is typically (conditionally) i.i.d., i.e., $\rho(x_{1:n}|y_{1:n}) = \prod_{t=1}^{n} \rho(x_t|y_t)$, which is chronological, but note that the Bayes mixture ξ is *not* i.i.d. One can show that the class of lower semi-computable chronological semimeasures $\mathcal{M}_U^{|} = \{\nu_1(\cdot|\cdot), \nu_2(\cdot|\cdot), \ldots\}$ is effectively enumerable.

The generalized universal a priori semimeasure also has two equivalent definitions:

$$M(x_{1:n}|y_{1:n}) := \sum_{p:U(p,y_{1:n})=x_{1:n}} 2^{-\ell(p)}$$

$$= \sum_{\nu \in \mathcal{M}} 2^{-K(\nu)} \nu(x_{1:n}|y_{1:n}) \qquad (11)$$

which is again in $\mathcal{M}_U^{|}$. In case of $|\mathcal{Y}| = 1$, this reduces to (1) and (5). The bounds (6) and (10) and others continue to hold, now for all individual y's, i.e., M predicts asymptotically x_t from y_t and $(y_{<t}, x_{<t})$ for *any* y, provided x is sampled from a computable probability measure $\mu(\cdot|y_{1:\infty})$. Convergence is rapid if μ is not too complex.

Universal reinforcement learning. The generalized universal a priori semimeasure (11) can be used to construct a universal reinforcement learning agent, called AIXI. In reinforcement learning, an *agent* interacts with an *environment* in cycles $t = 1, 2, \ldots, n$. In cycle t, the agent chooses an *action* y_t (e.g., a limb movement) based on past *perceptions* $x_{<t}$ and past actions $y_{<t}$. Thereafter, the agent perceives $x_t \equiv o_t r_t$, which consists of a (regular) *observation* o_t (e.g., a camera image) and a real-valued *reward* r_t. The reward may be scarce (e.g., just +1 (-1) for winning (losing) a chess game and 0 at all other times), internal (e.g., a robot's battery level), external (provided by a teacher), or universal (Orseau et al. 2013). Then the next cycle $t + 1$ starts. The goal of the agent is to maximize its expected reward over its lifetime n. Probabilistic planning deals with the situation in which the environmental probability distribution $\mu(x_{1:n}|y_{1:n})$ is known. Reinforcement learning deals with the case of unknown μ. In universal reinforcement learning, the unknown μ is replaced by M similarly to the prediction, decision, and classification cases above. The universally optimal action in cycle t is Hutter (2005, 2012)

$$y_t := \arg\max_{y_t} \sum_{x_t} \ldots \max_{y_n} \sum_{x_n} [r_t + \ldots + r_n] M(x_{1:n}|y_{1:n}) \qquad (12)$$

The expectations (Σ) and maximizations (max) over future x and y are interleaved in chronological order to form an expectimax tree similarly to minimax decision trees in extensive zero-sum games such as chess. Optimality and universality results similar to the prediction case exist.

Approximations and practical applications. Since K and M are only semi-computable, they have to be approximated in practice. For instance, $-\log M(x) = K(x) + O(\log \ell(x))$, and $K(x)$ can and has been approximated by off-the-shelf compressors such as Lempel-Ziv and successfully applied to a plethora of clustering problems (Cilibrasi and Vitányi 2005). The approximations upper-bound $K(x)$ and, e.g., for Lempel-Ziv converge to $K(x)$ if x is sampled from a context tree source. The ▶ minimum message length principle also attempts to approximate $K(x)$ for stochastic x. The context tree weighting algorithm considers a relatively large subclass of $\mathcal{M}_U$ that can be summed over efficiently. This can and has been combined with Monte Carlo sampling to efficiently approximate AIXI (12) (Veness et al. 2011). The time-bounded versions of K and M, namely, Levin complexity Kt and the speed prior S, have also been applied to various learning tasks (Gaglio 2007).

Other applications. Continuously parameterized model classes are very common in statistics. Bayesians usually assume a prior *density* over some parameter $\theta \in \mathbb{R}^d$, which works fine for many problems, but has its problems. Even for continuous classes $\mathcal{M}$, one can assign a (proper) universal prior (not density) $w_\theta^U := 2^{-K(\theta)} > 0$ for computable θ (and ν_θ) and 0 for uncomputable ones. This effectively reduces $\mathcal{M}$ to a discrete class $\{\nu_\theta \in \mathcal{M} : w_\theta^U > 0\} \subseteq \mathcal{M}_U$ which is typically dense in $\mathcal{M}$. There are various fundamental philosophical and statistical problems and paradoxes around (Bayesian) induction, which nicely disappear in the universal framework. For instance, universal induction has no zero and no improper p(oste)rior problem, i.e., can confirm universally quantified hypotheses, is reparametrization and representation invariant, and avoids the old evidence and updating problem, in contrast to most classical continuous prior densities. It even performs well in incomputable environments, actually better than latter (Rathmanner and Hutter 2011).

Discussion and Future Directions

Universal learning is designed to work for a wide range of problems without any a priori knowledge. In practice we often have extra information about the problem at hand, which could and should be used to guide the forecasting. One can incorporate it by explicating all our prior knowledge z and place it on an extra input tape of our universal Turing machine U or prefix our observation sequence x by z and use $M(zx)$ for prediction.

Another concern is the dependence of K and M on U. The good news is that a change of U

changes $K(x)$ only within an additive and $M(x)$ within a multiplicative constant independent of x. This makes the theory practically immune to any “reasonable” choice of U for large data sets x, but predictions for short sequences (shorter than typical compiler lengths) can be arbitrary. One solution is to take into account our (whole) scientific prior knowledge z (Hutter 2006), and predicting the now long string zx leads to good (less sensitive to “reasonable” U) predictions. This is a kind of grand transfer learning scheme. It is unclear whether a more elegant theoretical solution is possible.

Finally, the incomputability of K and M prevents a *direct* implementation of Solomonoff induction. Most fundamental theories have to be approximated for practical use, sometimes systematically such as polynomial time approximation algorithms or numerical integration, and sometimes heuristically like in many AI search problems or in non-convex optimization problems. Universal machine learning is similar, except that its core quantities are only semi-computable. This makes them often hard, but as described in the previous section, not impossible, to approximate.

In any case, universal induction can serve as a “gold standard” which practitioners can aim at. Solomonoff’s theory considers the class of all computable (stochastic) models, and a universal prior inspired by Ockham and Epicurus, quantified by Kolmogorov complexity. This led to a universal theory of induction, prediction, and decisions and, by including Bellman, to universal actions in reactive environments. Future progress on the issues above (incorporating prior knowledge, getting rid of the compiler constants, and finding better approximations) will lead to new insights and will continually increase the number of applications.

Cross-References

▶ Bayes’ Rule
▶ Bayesian Methods
▶ Bayesian Reinforcement Learning
▶ Classification
▶ Inductive Inference
▶ Loss
▶ Minimum Message Length
▶ Online Learning
▶ Prior Probability
▶ Classification
▶ Time Series

Optional Cross-References

▶ Data Set
▶ Discriminative Learning
▶ Hypothesis Space
▶ Induction
▶ Margin
▶ Minimum Description Length Principle
▶ Reinforcement Learning

Recommended Reading

Cilibrasi R, Vitányi PMB (2005) Clustering by compression. IEEE Trans Inf Theory 51(4):1523–1545

Gaglio M (2007) Universal search. Scholarpedia 2(11):2575

Hutter M (2005) Universal Artificial Intelligence: Sequential Decisions Based on Algorithmic Probability. Springer, Berlin

Hutter M (2006) Human knowledge compression prize. Open ended, http://prize.hutter1.net/

Hutter M (2007) On universal prediction and Bayesian confirmation. Theor Comput Sci 384(1):33–48

Hutter M (2012) One decade of universal artificial intelligence. In: Wang P, Goertzel B (eds) Theoretical Foundations of Artificial General Intelligence. Atlantis Press, Amsterdam, pp 67–88

Lattimore T, Hutter M (2011) No free lunch versus Occam’s razor in supervised learning. In: Proceedings of the Solomonoff 85th memorial conference, Melbourne. Volume 7070 of LNAI. Springer, pp 223–235

Li M, Vitányi PMB (2008) An Introduction to Kolmogorov Complexity and Its Applications, 3rd edn. Springer, Berlin

Orseau L, Lattimore T, Hutter M (2013) Universal knowledge-seeking agents for stochastic environments. In: Proceedings of the 24th international conference on algorithmic learning theory (ALT’13), Singapore. Volume 8139 of LNAI. Springer, pp 158–172

Rathmanner S, Hutter M (2011) A philosophical treatise of universal induction. Entropy 13(6):1076–1136

Schmidhuber J (2002) Hierarchies of generalized Kolmogorov complexities and nonenumerable universal measures computable in the limit. Int J Found Comput Sci 13(4):587–612

U

Solomonoff RJ (1964) A formal theory of inductive inference: Parts 1 and 2. Inf Control 7:1–22 and 224–254

Solomonoff RJ (1978) Complexity-based induction systems: comparisons and convergence theorems. IEEE Trans Inf Theory IT-24:422–432

Veness J, Ng KS, Hutter M, Uther W, Silver D (2011) A Monte-Carlo AIXI approximation. J Artif Intell Res 40:95–142

Wood I, Sunehag P, Hutter M (2011) (Non-)equivalence of universal priors. In: Proceedings of the Solomonoff 85th memorial conference, Melbourne. Volume 7070 of LNAI. Springer, pp 417–425

Unknown Attribute Values

▶ Missing Attribute Values

Unknown Values

▶ Missing Attribute Values

Unlabeled Data

Unlabeled data are data for which there are no target values. Unlabeled data are used in ▶ unsupervised learning. They stand in contrast to *labeled data* that have target values and are used in ▶ supervised learning.

Unsolicited Commercial Email Filtering

▶ Text Mining for Spam Filtering

Unstable Learner

An *unstable learner* produces large differences in generalization patterns when small changes are made to its initial conditions. The obvious initial condition is the set of training data used – for an unstable learner, sampling a slightly different training set produces a large difference in testing behavior. Some models can be unstable in additional ways, for example ▶ neural networks are unstable with respect to the initial weights. In general this is an undesirable property – high sensitivity to training conditions is also known as high variance, which results in higher overall mean squared error. The flexibility enabled by being sensitive to data can thus be a blessing or a curse. Unstable learners can however be used to an advantage in ▶ ensemble learning methods, where large variance is "averaged out" across multiple learners.

Examples of unstable learners are: neural networks (assuming gradient descent learning), and ▶ decision trees. Examples of stable learners are ▶ support vector machines, K-nearest neighbor classifiers, and ▶ decision stumps. It should of course be recognized that there is a continuum between "stable" and "unstable," and the opinion of whether something is "sensitive" to initial conditions is somewhat of a subjective one. See also ▶ bias-variance decomposition for a more formal interpretation of this concept.

Unsupervised Learning

Unsupervised learning refers to any machine learning process that seeks to learn structure in the absence of either an identified output (cf. ▶ supervised learning) or feedback (cf. ▶ reinforcement learning). Three typical examples of unsupervised learning are ▶ clustering, ▶ association rules, and ▶ self-organizing maps.

Uplift Modeling

Szymon Jaroszewicz
Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

Abstract

Uplift modeling is a machine learning technique which aims at predicting, on the level of

individuals, the gain from performing a given action with respect to refraining from taking it. Examples include medical treatments and direct marketing campaigns where the rate of spontaneous recovery and the background purchase rate need to be taken into account to assess the true gains from taking an action. Uplift modeling addresses this problem by using two training sets: the treatment dataset containing data on objects on which the action has been taken and the control dataset containing data on objects left untreated. A model is then built which predicts the difference between outcomes after treatment and without it conditional on available predictor variables.

An obvious approach to uplift modeling is to build two separate models on both training sets and subtract their predictions. In many cases, better results can be obtained with models which predict the difference in outcomes directly. A popular class of uplift models are decision trees with splitting criteria favoring tests which promote differences between treatment and control groups. Ensemble methods have proven to be particularly useful in uplift modeling, often leading to significant increases in performance over the base learners. Linear models, such as logistic regression and support vector machines, have also been adapted to this setting.

Dedicated methods, such as uplift or qini curves, are necessary for evaluating uplift models. Application of the methodology to survival data and scenarios with more than one possible action have also been considered.

Synonyms

Differential prediction; Net lift modeling; True lift modeling

Definition

Uplift modeling is a machine learning technique which aims at predicting, on the level of individuals, the gain from performing a given action such as a medical treatment or a direct marketing campaign. The learning task is to predict the difference between the outcome after applying the action to an individual and the outcome had the individual not been subjected to the action. The difficulty lies in the fact that the effects of the action cannot be reversed, so only one of those outcomes can be observed. Solving this task requires the use two training sets: the *experimental* or *treatment* group containing data on objects on which the action has been taken and the *control* group containing data on objects on which the action has not been taken. When the assignment to control and treatment groups is random, an uplift model assumes a *causal* interpretation and allows for selecting individuals to whom the action should be applied to achieve real benefits, for example, customers who will buy a product after a campaign but would not have bought it otherwise or patients who will recover after the treatment but will not be harmed by its side effects and would not have recovered spontaneously.

Introduction

Consider an application of a machine learning model to select customers for a direct marketing campaign. Typically, a small pilot campaign is conducted, and its results are used as training data to build the model. This approach, although frequently used in practice, is not correct. There are in fact four groups of customers:

1. Customers who bought *because* of the campaign, that is, bought after receiving the offer but would not have bought otherwise
2. Customers who bought the product but would have bought it even without the campaign
3. Customers who did not buy the product and the campaign had no effect
4. Customers who were originally going to buy the product but were put off by the campaign

Only customers in the first group should become targets of the marketing action. Targeting the two following groups only generates unnecessary costs and targeting the fourth group is outright harmful. The existence of the fourth group may seem counterintuitive, but is a well-known phenomenon in marketing. Similar groups can be identified in case of medical therapies: we only want to treat patients who recover after receiving the treatment and would not have recovered spontaneously. The fourth group in the above list corresponds to patients who would have recovered without the treatment but were, instead, harmed by its side effects.

The proper way to address the problem of selecting targets is to take into consideration the outcomes both after taking the action and after refraining from it. A typical approach is to use an additional—*control*—training set containing outcomes for individuals who where put aside and not subjected to the action. The other dataset, with outcomes for individuals subjected to the action is called the *experimental* or *treatment* training set.

The goal of *uplift modeling* is to build a model which predicts the difference between outcomes after taking the action and after refraining from taking it, based on the two training sets. For example, in case of a binary target variable Y, with the outcome $Y = 1$ interpreted as success, the quantity predicted by an uplift model typically is the difference between success probabilities in the two groups:

$$P(Y=1|x, \text{action applied}) - P(Y=1|x, \text{control}), \quad (1)$$

where x is a feature vector describing the object for which prediction is being made. This quantity is referred to as *net gain*, *true gain*, or *uplift*. Analogues for regression problems and modifications involving costs also exist (Holland 1986; Hansotia and Rukstales 2002).

The main algorithmic difficulty lies in the fact that for each individual only one of those outcomes is known, never both. This problem is called *fundamental problem of causal inference* (Holland 1986). As a result the predicted target value is not known at the level of individual data records, which has implications for the design of learning algorithms as well as for model quality assessment.

Structure of the Learning System

Uplift modeling works on two training sets: *treatment* and *control*, so uplift learning algorithms need to take into account the additional control dataset. Below we describe several types of uplift learners, most of which originate from corresponding classification algorithms.

The Two-Model Approach

An obvious approach to uplift modeling is the *two-model approach*. Separate probabilistic classifiers are built on the treatment and control datasets, and their predictions are subtracted to obtain an estimate of the net gain given in Eq. 1.

A clear advantage of the two-model approach is its intuitive clarity and simplicity. However, in many cases, its performance can be poor because both models try to predict the value of the target variable in the two training sets instead of focusing on the (usually quite small) differences in behavior between the two groups. An intuitive example is given in Radcliffe and Surry (2011) where an artificial dataset is constructed with one variable strongly influencing the outcome in both groups and a second variable weakly influencing the outcome in the treatment group only. A model based on two separate decision trees was then built on the data. Both trees split only based on the first variable, leading to a useless uplift model. A single decision tree built to directly predict the net gain would select tests based on the second variable, at least in the upper levels of the tree, leading to a much better performance.

Most research on uplift modeling has therefore concentrated on building models predicting the net gain *directly*.

Uplift Decision Trees

Decision trees were, historically, the first learning algorithms adapted the problem of uplift modeling. The term "uplift" first appeared in a machine learning context in Radcliffe and Surry (1999), where differential response trees were described, albeit with little technical detail given. A detailed description was later provided in Radcliffe and Surry (2011). The algorithm uses a special splitting criterion designed to pick tests which promote differences between success rates in the treatment and control groups. The criterion works by first constructing a linear model with an interaction term between the split outcome and the group to which data records belong (treatment or control). The coefficient estimate for this interaction term corresponds to the estimated difference between treatment and control group behavior on both sides of the split, and the quality of the split is measured by the p-value of a statistical test for this coefficient.

Uplift decision trees based on information theoretical criteria have been proposed in Rzepakowski and Jaroszewicz (2010, 2012). The difference in class distributions in the treatment and control groups is measured using a statistical divergence measure such as the Kullback-Leibler divergence. Splits are then selected which result in the highest increase of the divergence. The authors show that the proposed criteria possess several desirable properties such as not selecting tests independent from the outcome variable or reducing to standard decision tree criteria when the control group is absent. A dedicated pruning method is also provided. An advantage of this approach is that it is able to handle target variables with more than two outcomes.

Ensemble Methods

In classical machine learning, one way to improve performance of decision trees is to combine them into ensembles. This often results in significant increases in accuracy over the base learners. Ensemble methods have also been successfully applied in uplift modeling.

The first mention of the use of uplift bagging ensembles can be found in Radcliffe and Surry (2011) but little detail is given. Guelman et al. (2012) devised an uplift random forest by using additional randomization in the decision tree test selection based on the criterion given in Rzepakowski and Jaroszewicz (2012).

In Sołtys et al. (2015) a thorough experimental analysis of bagging and random forests in uplift modeling was conducted showing that those methods often dramatically improve the performance of base learners; moreover, the effect is more significant than for classification. Some theoretical justification has been offered: frequently, the class variable in both groups is strongly influenced by predictor variables, but the differences between the treatment and control groups are quite small. Modeling this small differences is difficult and leads to uplift trees with highly randomized structure and, as a consequence, to highly diverse ensembles.

Regression Methods

Linear regression techniques are a very important tool of predictive analytics, so there have been several attempts to apply them to uplift modeling. Many approaches are in fact variations on the two-model solution. For example, in medical applications, a model with interactions between predictors and the action indicator is often used:

$$y = \alpha' x + \beta' x A,$$

where A is an indicator variable taking the value of 1 if the action has been applied to a given individual and the value of 0 otherwise, x is the vector of input predictors (including a constant intercept term), α and β are regression coefficient vectors, and the prime denotes matrix transpose. The expression $\beta' x$ is the predicted net gain. A similar approach has been presented in Lo (2002); however, two additional steps of variable selection are used.

A method based on a class variable transformation which allows for building a single uplift regression model has been presented in Jaśkowski and Jaroszewicz (2012). The idea is to use a new target variable Z defined as follows:

$$Z = \begin{cases} Y & \text{if } A = 1, \\ 1 - Y & \text{if } A = 0. \end{cases}$$

In other words, the class variable in the control group is reversed, and both training sets are concatenated into a single dataset. It can be shown that modeling the probability of the event $Z = 1$ is equivalent to modeling the net gain given in Eq. 1. The transformation can thus be used to turn any probabilistic classifier into an uplift model. The idea has been presented earlier in Lai et al. (2006), but without a formal justification. In Jaśkowski and Jaroszewicz (2012) the approach has been used with a logistic regression model, but the benefits with respect to an approach based on subtracting predictions of two separate logistic models are less clear than in the case of decision trees.

Another technique, called *g-estimation*, which has been presented in medical statistics literature (Robins 1994), can be interpreted as an earlier approach to uplift regression. The technique is based on selecting a vector of uplift model coefficients which make the expectations of the target variable in the treatment and control groups equal. Asymptotic optimality results are provided but require correct specification of outcomes in the control group. Due to lack of publicly available implementations, the method has not yet gained significant popularity.

Other Algorithms and Extensions

Several other uplift modeling algorithms have been proposed. Two variants of uplift support vector machines have been developed in Zaniewicz and Jaroszewicz (2013) and Kuusisto et al. (2014). The first approach uses two parallel separating hyperplanes which split points into three groups with predicted positive, neutral, and negative impact of the action. The second approach uses a modified SVM to directly maximize the area under the uplift curve (see below). Other methods, such as k-nearest neighbors and naive Bayesian classifier, have been adapted to the uplift setting by Kim Larsen in a series of conference talks (Larsen 2011).

In many applications, uplift modeling needs to be applied to survival data which involves censoring. For example, if a patient entered a medical study a year ago and is still alive, we do not know her survival time; all we know is that it is at least one year. The true value has been *censored*. Typically, censored data requires special methods, but in Jaroszewicz and Rzepakowski (2014) it has been shown that in case of uplift modeling such data can, under certain assumptions, be converted into a problem with a binary class. Cases for which the observed (possibly censored) survival time exceeds some threshold are considered successes. The resulting uplift model cannot predict the true gain in survival rate but can make a correct decision whether the action should or should not be applied to a given individual.

In real-world applications, more than one action is often possible. For example, we may contact a customer using a text message, regular mail, or a phone call. The task now is not only to decide whether a given individual should be subjected to an action but also to select the most profitable action to take. The two-model approach can easily be extended to multiple actions. In Rzepakowski and Jaroszewicz (2012) direct extension of uplift decision trees to the multiple actions scenario is provided.

Evaluation of Uplift Models

Due to the fundamental problem of causal inference, we can never determine whether the action was truly beneficial for a given individual; such assessment can only be made on the level of groups of individuals. Moreover, two test sets are now necessary: treatment (test data on objects subjected to the action) and control (test data on objects left untreated).

A typical approach scores both test sets using the uplift model being evaluated and groups the test cases by deciles of the score. The success rate in the first decile of the control test set is subtracted from the success rate in the first decile of the treatment test set. This way, an estimate of model performance for individuals whose score was in the top 10 % is obtained. Analogous estimates are computed for the remaining deciles and tabulated (Hansotia and Rukstales 2002).

A more convenient approach is to visualize uplift model's performance using curves. A starting point is *cumulative gains curves* (also known as *lift curves*) drawn on the treatment and control training sets. The x-axis of a cumulative gains curve denotes the percentage of the population assigned highest scores by a model and the y-axis the success rate after targeting the given percentage of the population. Cumulative gains curves drawn on the treatment and control test sets are subtracted to obtain a single curve describing the performance of an uplift model. Such curves have been called *uplift curves* in Rzepakowski and Jaroszewicz (2012) and *incremental gains curves* or *qini curves* in Radcliffe and Surry (2011). The curves are a very convenient tool in campaign planning: the x-axis corresponds to the size of the campaign, and the y-axis provides an estimate of the resulting total net gain.

Applications

Uplift modeling is gaining importance primarily in predictive analytics and direct marketing. Several successful applications in the banking and telecommunication sectors are reported in literature (Radcliffe and Surry 2011; Siegel and Davenport 2013). Uplift modeling played a crucial role in Barak Obama's 2012 presidential campaign (Siegel and Davenport 2013). A potentially large application area is personalized medicine where appropriate treatment is selected based on patient's individual characteristics.

Cross-References

- ▶ Causal Discovery
- ▶ Online Controlled Experiments and A/B Testing

Recommended Reading

Guelman L, Guillén M, Pérez-Marín AM (2012) Random forests for uplift modeling: an insurance customer retention case. In: Modeling and simulation in engineering, economics and management. Lecture notes in business information processing (LNBIP), vol 115. Springer, Heidelberg, pp 123–133

Hansotia B, Rukstales B (2002) Incremental value modeling. J. Interact Mark 16(3):35–46

Holland PW (1986) Statistics and causal inference. J Am Stat Assoc 81(396):945–960

Jaroszewicz S, Rzepakowski P (2014) Uplift modeling with survival data. In: ACM SIGKDD workshop on health informatics (HI-KDD'14), New York

Jaśkowski M, Jaroszewicz S (2012) Uplift modeling for clinical trial data. In: ICML 2012 workshop on machine learning for clinical data analysis, Edinburgh

Kuusisto F, Santos Costa V, Nassif H, Burnside E, Page D, Shavlik J (2014) Support vector machines for differential prediction. In: ECML-PKDD, Nancy

Lai Y-T, Wang K, Ling D, Shi H, Zhang J (2006) Direct marketing when there are voluntary buyers. In: Sixth International Conference on Data Mining, 2006 (ICDM'06), IEEE, Los Alamitos, pp 922–927. http://www.comp.hkbu.edu.hk/iwi06/icdm/

Larsen K (2011) Net lift models: optimizing the impact of your marketing. In: Predictive analytics world, workshop presentation, San Francisco

Lo VSY (2002) The true lift model—a novel data mining approach to response modeling in database marketing. SIGKDD Explor 4(2):78–86

Radcliffe NJ, Surry PD (1999) Differential response analysis: modeling true response by isolating the effect of a single action. In: Proceedings of credit scoring and credit control VI. Credit Research Centre, University of Edinburgh Management School

Radcliffe NJ, Surry PD (2011) Real-world uplift modelling with significance-based uplift trees. Portrait Technical Report TR-2011-1, Stochastic Solutions

Robins J (1994) Correcting for non-compliance in randomized trials using structural nested mean models. Commun Stat—Theory Methods 23(8):2379–2412

Rzepakowski P, Jaroszewicz S (2010) Decision trees for uplift modeling. In: Proceedings of the 10th IEEE international conference on data mining (ICDM), Sydney, pp 441–450

Rzepakowski P, Jaroszewicz S (2012) Decision trees for uplift modeling with single and multiple treatments. Knowl Inf Syst 32:303–327

Siegel E, Davenport TH (2013) Predictive analytics: the power to predict who will click, buy, lie, or die. Wiley, Hoboken

Sołtys M, Jaroszewicz S, Rzepakowski P (2015) Ensemble methods for uplift modeling. Data Min Knowl Discov 29(6):1531–1559

Zaniewicz Ł, Jaroszewicz S (2013) Support vector machines for uplift modeling. In: The first IEEE ICDM workshop on causal discovery (CD 2013), Dallas

Utility Problem

▶ Explanation-Based Learning

V

Value Function Approximation

Michail G. Lagoudakis
Technical University of Crete, Chania,
Greece

Abstract

The goal in sequential decision making under uncertainty is to find good or optimal policies for selecting actions in stochastic environments in order to achieve a long-term goal; such problems are typically modeled as Markov decision processes (MDPs). A key concept in MDPs is the *value function*, a real-valued function that summarizes the long-term goodness of a decision into a single number and allows the formulation of optimal decision making as an optimization problem. An exact representation of value functions in large real-world problems is infeasible; therefore, a large body of research has been devoted to *value-function approximation* methods, which sacrifice some representation accuracy for the sake of scalability. These approaches have delivered effective approaches to deriving good policies in hard decision problems and laid the foundation for efficient reinforcement learning algorithms, which learn good policies in unknown stochastic environments through interaction.

Synonyms

Approximate dynamic programming; Cost-to-go function approximation; Neuro-dynamic programming

Definition

Value Function Approximation is a collection of function approximation representations, techniques, and methods aiming at providing a scalable and effective approximation to an exact value function (a real-valued function indicating the long-term goodness of making decisions at any state within a sequential decision problem).

Motivation and Background

Markov Decision Processes

A *Markov decision process* (MDP) is a 6-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mathcal{D})$, where $\mathcal{S}$ is the state space of the process, $\mathcal{A}$ is a finite set of actions, $\mathcal{P}$ is a Markovian transition model ($\mathcal{P}(s'|s,a)$ denotes the probability of a transition to state s' when taking action a in state s), $\mathcal{R}$ is a reward function ($\mathcal{R}(s,a)$ is the reward for taking action a in state s), $\gamma \in (0,1]$ is the discount factor for future rewards (a reward received after t steps is weighted by γ^t), and $\mathcal{D}$ is the initial state distribution (Puterman 1994). MDPs are discrete-time processes. The process begins at time $t = 0$

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

in some state $s_0 \in \mathcal{S}$ drawn from $\mathcal{D}$. At each time step t, the decision maker observes the current state of the process $s_t \in \mathcal{S}$ and chooses an action $a_t \in \mathcal{A}$. The next state of the process s_{t+1} is drawn stochastically according to the transition model $\mathcal{P}(s_{t+1}|s_t, a_t)$, and the reward r_t at that time step is determined by the reward function $\mathcal{R}(s_t, a_t)$. The horizon h is the temporal extent of each run of the process and is typically infinite. A complete run of the process over its horizon is called an *episode* and consists of a long sequence of states, actions, and rewards:

$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} s_3 \xrightarrow[r_3]{a_3} s_4 \quad \ldots \quad s_{h-1} \xrightarrow[r_{h-1}]{a_{h-1}} s_h .$$

The quantity of interest is the *expected total discounted reward* from any state s:

$$E\Big(r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \cdots + \gamma^h r_h \;\Big|\; s_0 = s\Big) = E\left(\sum_{t=0}^{h} \gamma^t r_t \;\middle|\; s_0 = s\right),$$

where the expectation is taken with respect to all possible trajectories of the process in the state space under the decisions made and the transition model, assuming that the process is initialized in state s. The goal of the decision maker is to make decisions so that the expected total discounted reward, when s is drawn from $\mathcal{D}$, is optimized. (The optimization objective could be maximization or minimization depending on the problem. Here, we adopt a reward maximization viewpoint, but there are analogous definitions for cost minimization. There are also other popular optimality measures, such as maximization/minimization of the average reward/cost per step.)

Policies

A *policy* dictates how the decision maker chooses actions in each state. A *stationary, deterministic policy* π is a mapping $\pi : \mathcal{S} \mapsto \mathcal{A}$ from states to actions; $\pi(s)$ denotes the action the agent takes in state s. In this case, there is a single action choice for each state, and this choice does not change over time. In contrast, a *stationary, stochastic policy* π is a mapping $\pi : \mathcal{S} \mapsto \Omega(\mathcal{A})$, where $\Omega(\mathcal{A})$ is the set of all probability distributions over $\mathcal{A}$. Stochastic policies are also called *soft*, for they do not commit to a single action per state; $\pi(a|s)$ stands for the probability of choosing action a in state s under policy π. Each policy π (deterministic or stochastic) is characterized by the expected total discounted reward it accumulates during an episode. An *optimal policy* π^* for an MDP is a policy that maximizes the expected total discounted reward from any state $s \in \mathcal{S}$. It is well known that for every MDP, there exists at least one, not necessarily unique, optimal policy, which is stationary and deterministic.

Value Functions

The quality of any policy π can be quantified formally through a value function, which measures the expected return of the policy under different process initializations. For any MDP and any policy π, the *state value function* V assigns a numeric value to each state. The value $V^\pi(s)$ of a state s under a policy π is the expected return, when the process starts in state s and the decision maker follows policy π (all decisions at all steps are made according to π):

$$V^\pi(s) = E_{a_t \sim \pi\,;\, s_t \sim \mathcal{P}\,;\, r_t \sim \mathcal{R}} \left(\sum_{t=0}^{\infty} \gamma^t r_t \;\middle|\; s_0 = s\right).$$

Similarly, the *state-action value function* Q assigns a numeric value to each pair (s, a) of states and actions. The value $Q^\pi(s, a)$ of taking action a in state s under a policy π is the expected return when the process starts in state s, and the decision maker takes action a for the first step and follows policy π thereafter:

$$Q^{\pi}(s,a)$$

$$= E_{a_t \sim \pi\,;\, s_t \sim \mathcal{P}\,;\, r_t \sim \mathcal{R}} \left(\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, s_0 = s,\, a_0 = a \right).$$

The state and the state-action value functions for a deterministic policy π are related as follows:

$$V^{\pi}(s) = Q^{\pi}\big(s, \pi(s)\big).$$

For a stochastic policy π this relationship needs to take into account the probability distribution over actions:

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q^{\pi}(s,a).$$

The state-action value function of a policy π (either deterministic or stochastic) can also be expressed in terms of the state value function:

$$Q^{\pi}(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) V^{\pi}(s').$$

The *optimal value functions* $V^* = V^{\pi^*}$ and $Q^* = Q^{\pi^*}$ are the state and the state-action value functions of any optimal policy π^*. Even if there are several distinct optimal policies, they all share the same unique optimal value functions.

Bellman Equations

Given the full MDP model, the state or the state-action value function of any given policy can be computed by solving a linear system formed using the linear Bellman equations. In general, the linear *Bellman equation* relates the value of the function at any point to the values of the function at several – in fact, all – other points. This is achieved by separating the first step of an episode from the rest and using the definition of the value function recursively in the next step. In particular, for any deterministic policy π, the linear Bellman equation for the state value function is

$$V^{\pi}(s) = \mathcal{R}(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, \pi(s)) V^{\pi}(s'),$$

whereas for a stochastic policy π, it becomes

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \Big(\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) V^{\pi}(s') \Big).$$

The exact V^{π} values for all states can be found by solving the $(|\mathcal{S}| \times |\mathcal{S}|)$ linear system that results from writing down the linear Bellman equation for all states.

Similarly, the linear Bellman equation for the state-action value function given any deterministic policy π is

$$Q^{\pi}(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) Q^{\pi}\big(s', \pi(s')\big),$$

whereas for a stochastic policy π, it becomes

$$Q^{\pi}(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) \times \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^{\pi}(s',a').$$

The exact Q^{π} values for all state-action pairs can be found by solving the $(|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}||\mathcal{A}|)$ linear system that results from writing down the linear Bellman equation for all state-action pairs.

The unique optimal state or state-action value function can be computed even for an unknown optimal policy π^* using the nonlinear *Bellman optimality equation*, which relates values of the function at different points while exploiting the fact that there exists a deterministic optimal policy that achieves the maximum value at each point. In particular, the nonlinear Bellman optimality equation for the state value function is

$$V^*(s) = \max_{a \in \mathcal{A}} \Big\{ \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) V^*(s') \Big\},$$

whereas for the state-action value function is

$$Q^*(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) \max_{a' \in \mathcal{A}} \left\{ Q^*(s',a') \right\}.$$

The functions V^* and Q^* can be approximated arbitrarily closely by the iterative application of the operator formed by the right-hand side of the equations above (Bellman optimality operator). This iteration is a contraction with rate γ, so starting with any arbitrary initialization, it eventually converges to V^* or Q^*.

Significance of Value Functions

Value functions play a critical role in sequential decision making because they address two core problems: policy evaluation and policy improvement. Policy evaluation refers to the problem of quantifying the quality of any given policy π in a given MDP. Apparently, computing the value function V^π or Q^π using the Bellman equations provides the solution to this problem. Policy improvement, on the other hand, refers to the problem of deriving an improved policy π' given any base policy π, so that π' is at least as good as π and possibly better. The knowledge of V^π or Q^π allows for the derivation of an improved deterministic policy π' through a simple one-step look-ahead maximization procedure:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} \left\{ \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) V^\pi(s') \right\}$$

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} \left\{ Q^\pi(s,a) \right\}.$$

Note that this maximization does not need the MDP model when using the state-action value function. Once policy evaluation and policy improvement have been addressed, the derivation of an optimal policy for any MDP is straightforward. One can alternate between policy evaluation and policy improvement producing a sequence of improving policies until convergence to an optimal policy; this algorithm is known as policy iteration. Alternatively, one can iteratively compute an optimal value function V^* or Q^* and extract an optimal policy through a trivial step of policy improvement on top of V^* or Q^*; this algorithm is known as value iteration. In either case, value functions provide the means to the end.

The problem of deriving an optimal policy using the full MDP model is known as planning. Nevertheless, in many real-world sequential decision domains, the model is unknown. The problem of optimal decision making in an unknown stochastic environment is known as reinforcement learning, because the decision maker relies on the feedback received through interaction with the environment to reinforce or discourage past decisions. More specifically, the learner interacts with an unknown MDP and typically observes the state of the process and the immediate reward at every step; however, $\mathcal{P}$ and $\mathcal{R}$ are not accessible. At each step of interaction, the learner observes the current state s, chooses an action a, and observes the resulting next state s' and the reward received r, thus learning is based on (s,a,r,s') samples. The core problems in reinforcement learning are known as prediction and control. Prediction refers to the problem of learning the value function of a given policy π in an unknown MDP through interaction. Well-known algorithms for the prediction problem are Monte Carlo estimation and temporal difference (TD) learning. Control, on the other hand, refers to the problem of gradually learning a good or even optimal policy in an unknown MDP through interaction. Well-known algorithms for the control problem are SARSA and Q-learning. Again, value functions play a critical role in reinforcement learning; they are absolutely necessary for the prediction problem, and the majority of control approaches are value-function based.

Structure of Learning System

Value-Function Approximation

Most algorithms for planning or learning in MDPs rely on computing or learning a value function. However, if the state space of the process is fairly large, the exact (tabular) representation of a value function becomes problematic. Not only does memory space become insufficient very quickly, but also computing or learning accurately all the distinct entries of the function requires a tremendous amount of computation and data. This is known as the *curse of dimensionality*: the exponential growth of the state or action space as a function of the dimensionality of the state or action. The urgent need for solutions to large real-world sequential decision problems has drawn attention to approximate methods. These methods use function approximation techniques for approximating value functions; therefore, they sacrifice some representational accuracy in order to make the representation manageable in practice. Sacrificing accuracy in the representation of the value function is acceptable, since the ultimate goal is to find a good policy and not necessarily an accurate value function. As a result, value-function approximation methods cannot guarantee optimal solutions, but only good solutions. This is not to say that they are doomed to always finding suboptimal solutions; if an optimal solution lies within the space spanned by the value-function approximation scheme, it is possible that an optimal solution will be discovered.

Let $\hat{V}^{\pi}(s;w)$ be an approximation to the state value function $V^{\pi}(s)$ represented by a parametric approximation architecture with free parameters w. The key idea of value- function approximation is that the parameters w can be adjusted appropriately so that the approximate values are "close enough" to the original values,

$$\hat{V}^{\pi}(s;w) \approx V^{\pi}(s),$$

and, therefore, $\hat{V}^{\pi}$ can be used in place of the exact value function V^{π}. Similarly, let $\hat{Q}^{\pi}(s,a;w)$ be an approximation to the state-action value function $Q^{\pi}(s,a)$. Again, the goal is to adjust the parameters w so that

$$\hat{Q}^{\pi}(s,a;w) \approx Q^{\pi}(s,a),$$

and, therefore, $\hat{Q}^{\pi}$ can be used in place of the exact value function Q^{π}. Approximations $\hat{V}^{*}$ and $\hat{Q}^{*}$ of the optimal value functions V^{*} and Q^{*} are defined similarly. The characterization "close enough" ($\approx$) accepts a variety of interpretations in this context, and it does not necessarily refer to the minimization of some norm. Value-function approximation should be regarded as a *functional* approximation rather than as a pure *numerical* approximation, where "functional" refers to the ability of the approximation to play closely the functional role of the original value function within a decision making algorithm.

The benefits of value-function approximation are obvious. The storage requirements are much smaller compared to the tabular case, since only the parameters w need to be stored along with a compact description of the functional form of the architecture. In general, for most approximation architectures, the storage needs are independent of the size of the state space and/or the size of the action space. Furthermore, for most approximation architectures there is no restriction on the state space to be a finite set; it could be an infinite, or even a continuous, space. This flexibility nevertheless reveals the need for good generalization abilities on behalf of the architecture, since the approximate value function will have to provide good values over the entire state/state-action space, using data only from a limited subset of the space.

The main difficulty associated with value-function approximation, beyond the loss in accuracy, is the choice of the *projection method*, which is the method of finding appropriate parameters that maximize the accuracy of the approximation according to certain criteria and with respect to the target function. Typically, for ordinary function approximation, this is accomplished using a training set of examples of the form $\{s, V^{\pi}(s)\}$, $\{s, V^{*}(s)\}$, $\{(s,a), Q^{\pi}(s,a)\}$, or $\{(s,a), Q^{*}(s,a)\}$ that provide the true value of the target function at certain sample points s or (s,a) (supervised learning). Unfortunately, in the context of sequential decision making, the target value function is completely unknown. Had it been possible to compute it easily, value-function approximation would have been unnecessary. In fact, it is not possible to analytically compute the true value of the target value function even at certain isolated sample points due to interdependencies between the values at all points. The implication of this difficulty is that evaluation and projection to the approximation architecture must be blended together. This is usually achieved by trying to find values for the free parameters so that the approximate function retains some properties of the original exact value function. Another implication of using approximation for value functions is that all convergence properties of exact planning or learning algorithms are compromised. Therefore, significant attention must be paid to the choice of the approximation architecture and the evaluation and projection method to minimize the chances for divergence or oscillations.

Approximation Architectures

There are a variety of architectures available for value-function approximation: perceptrons, neural networks, splines, polynomials, radial basis functions, support vector machines, decision trees, CMACs, wavelets, etc. These architectures have diverse representational power and generalization abilities, and the most appropriate choice will heavily depend on the properties of the decision making problem at hand. The projection methods associated with these approximation architectures are typically designed for a supervised learning setting. For successful use in the context of decision making, combined evaluation and projection methods are necessary.

A broad categorization of approximation architectures distinguishes between nonlinear and linear architectures. The characterization "nonlinear" or "linear" refers to the way the free parameters enter into the architecture and not to the approximation ability of the architecture. Nonlinear architectures are usually more expressive than the linear ones, due to the complex interactions among their free parameters; however, tuning their parameters is a much more elaborate task compared to tuning the parameters of their linear counterparts. Linear architectures are perhaps the most popular choice for value-function approximation; interestingly, most theoretical results on convergence properties in the context of value-function approximation are restricted to linear architectures.

A *linear approximation architecture* approximates a function $V^{\pi}(s)$ or $Q^{\pi}(s,a)$ as a linear weighted combination of k *basis functions* (also called *features*):

$$\hat{V}^{\pi}(s;w) = \sum_{j=1}^{k} \phi_j(s) w_j = \phi(s)^{\top} w$$

$$\hat{Q}^{\pi}(s,a;w) = \sum_{j=1}^{k} \phi_j(s,a) w_j = \phi(s,a)^{\top} w.$$

The free *parameters* of the architecture are the coefficients w_j of the combination (also called *weights*). The basis functions ϕ_j are fixed, but arbitrary and, in general, nonlinear functions of s or (s,a). It is required that the basis functions ϕ_j are linearly independent to ensure that there are no redundant parameters and that the matrices involved in the computations are full rank. In general, $k \ll |\mathcal{S}|$ and $k \ll |\mathcal{S}||\mathcal{A}|$ and the basis functions ϕ_j have small compact descriptions. As a result, the storage requirements of a linear approximation architecture are much smaller than those of the tabular representation of a value function. There is a large variety of linear approximation architectures, and in fact, many common schemes for value-function approximation can be cast as linear architectures.

- *Lookup Table*. This is the exact tabular representation (There is no approximation under this scheme; it is included only to illustrate that exact representation belongs in the family of linear architectures.) suitable for problems with discrete state variables. Each basis function is an indicator function whose value is 1 only for a specific discrete input point (state or state-action) and 0 otherwise. Each parameter stores one value/entry of the table.
- *Discretization*. This is a common technique for turning a continuous space into discrete using a uniform- or variable-resolution grid. One indicator basis function is assigned to each cell of the discretization, and the corresponding parameter holds the value of that cell.

- *Tile Coding (CMAC)*. This scheme utilizes several overlapping discretizations (tilings) for better accuracy. It generates indicator basis functions for each cell of each tiling and concatenates the basis functions for all tilings. Each parameter corresponds to one cell in one tiling, but the value at each input point is computed additively from the values of all containing cells from all tilings.
- *State Aggregation*. This is a family of schemes where "similar" (by some metric) states are grouped together and are treated as one state. The similarity metric is usually formed through dimensionality reduction techniques for identifying the most significant dimensions in a high-dimensional input space, unlike conventional proximity measures in the same space. There is one indicator basis function for each group and a single value for all states in the group.
- *Polynomials*. This is a generic approximation scheme suitable for problems with several (continuous) state variables. Each basis function is a polynomial term composed of state variables up to a certain degree.
- *Radial Basis Functions (RBFs)*. This is another generic approximation scheme suitable for continuous state variables. Each basis function is a Gaussian with fixed mean and variance; the Gaussians are topologically arranged so that they cover the input space with some overlap.
- *Kernel Methods*. Kernels are symmetric functions between two points, and they are used to represent compactly dot products of feature vectors in high- or even infinite-dimensional spaces. The compactness of kernels allows for approximation schemes that essentially enjoy the flexibility provided by a huge or infinite number of basis functions. The basis functions, in this case, are implicitly defined through the choice of the kernel.
- *Partitioning*. This technique is used for constructing complex approximators by partitioning the state space in several subsets and using a different approximator in each one of them. If linear architectures are used in all partitions, then a set of basis functions for the global architecture can be constructed by concatenating the basis functions of the smaller linear architectures multiplying each subset with an indicator function for the corresponding partition.

Linear architectures offer several advantages: they are easy to implement and use, and their behavior is fairly transparent, both from an analysis standpoint and from a debugging and feature engineering standpoint. It is usually relatively easy to get some insight into the reasons for which a particular choice of features succeeds or fails. This is facilitated by the fact that the magnitude of each parameter is related to the importance of the corresponding feature in the approximation (assuming normalized features).

A *nonlinear approximation architecture* approximates a function $V^{\pi}(s)$ or $Q^{\pi}(s,a)$ using arbitrary parametric functions of s and (s,a), possibly in conjunction with some *features* ϕ computed over s and (s,a). The best-known representative of this category is the multilayer feed-forward neural networks, which use one or more layers of linear combinations followed by a nonlinear sigmoidal transformation (thresholding). In their simplest form (one layer), neural networks approximate value functions as

$$\begin{aligned}\hat{V}^{\pi}(s;w,\theta) &= \sum_{i=1}^{m}\theta_i\sigma\left(\sum_{j=1}^{k}\phi_j(s)w_{ji}\right)\\ &= \sum_{i=1}^{m}\theta_i\sigma\Big(\phi(s)^{\top}w_i\Big)\\ \hat{Q}^{\pi}(s,a;w,\theta) &= \sum_{i=1}^{m}\theta_i\sigma\left(\sum_{j=1}^{k}\phi_j(s,a)w_{ji}\right)\\ &= \sum_{i=1}^{m}\theta_i\sigma\Big(\phi(s,a)^{\top}w_i\Big).\end{aligned}$$

Common choices for the differentiable, bounded, and monotonically increasing function σ are the hyperbolic tangent function $\sigma(x) = \tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$ and the logistic function $\sigma(x) = 1/(1 + e^{-x})$. The free *parameters* of the architecture (also called *weights*) are the coefficients w_{ji} of the linear combinations of the inputs and the coefficients θ_i of the linear combination of the sigmoidal transformations for the output. Notice that the parameters w_{ji} enter nonlinearly into the approximation.

A key question in all approximation architectures is *how* features are generated and selected. The *feature generation and selection* problem is an open question that spans most of machine learning research and admits no easy and general answer. Prior domain-specific knowledge and experience can be very helpful in choosing appropriate features. Several recent studies also describe methods for automatically generating features targeted for value-function approximation (Menache et al. 2005; Mahadevan and Maggioni 2007; Parr et al. 2007).

Learning

Learning (or *training* or *parameter estimation*) in value-function approximation refers to parameter tuning methods that take as input a policy π, an approximation architecture for V^π/Q^π, and the full MDP model or samples of interaction with the process and output a set of parameters w^π such that $\hat{V}^\pi/\hat{Q}^\pi$ is a good approximation to V^π/Q^π. Learning also covers methods for the harder problem of taking an approximation architecture for V^*/Q^* and the model or samples and outputting a set of parameters w^* such that $\hat{V}^*/\hat{Q}^*$ is a good approximation to V^*/Q^*. The former problem is somewhat easier because the policy π, unlike an optimal policy π^*, is known, and therefore in the presence of a simulator of the process, the value function can be estimated at any isolated point using Monte Carlo estimation techniques based on repeated policy rollouts from that point. Each rollout is an execution of an episode starting from a state s (or state-action (s,a)) using policy π to obtain an unbiased estimate of the return of the policy from s (or (s,a)). In this case, value-function approximation can be cast as a classic supervised learning problem; the true value of V^π/Q^π is estimated at a subset of points to form a training set, which can be subsequently used to train the approximation architecture using supervised learning techniques. However, in the absence of a simulator or when seeking approximations to V^*/Q^*, evaluation and projection into the architecture have to be carried out simultaneously.

The true value function has two key properties: it satisfies the Bellman equations, and it is the fixed point of the Bellman operator. Learning in value-function approximation strives to find values for the free parameters so that the approximate function retains at least one of these properties to the extent this is possible. Learning methods that focus on satisfying the Bellman equations attempt to find an approximate function that minimizes the Bellman residual, the difference between the left- and the right-hand sides of the system of Bellman equations. On the other hand, learning methods that focus on the fixed point property attempt to find an approximate function that exhibits a fixed point behavior under the combined application of the Bellman operator and projection onto the space spanned by the basis functions. Experimental evidence suggests that it is really hard to satisfy both properties under approximation, and therefore these two approaches differ significantly in their solutions. The majority of existing learning methods focus on fixed point approximation, which experimentally has been shown to exhibit more stable behavior and delivers better policies. There are also proposals for combining the benefits of both approaches into a hybrid method (Johns et al. 2009).

The most widely used learning approach is based on gradient descent and is applicable to any approximation architecture that is differentiable with respect to its parameters. Any stochastic approximation learning method for tabular representations of value functions can be extended to approximate representations. For example, given any sample (s,a,r,s') of interaction with the process, the temporal difference (TD) learning update rule

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha\Big(r + \gamma V^\pi(s') - V^\pi(s)\Big)$$

for tabular representations, where $\alpha \in (0,1]$ is the learning rate, becomes

$$w^{\pi} \leftarrow w^{\pi} + \alpha\Big(r + \gamma \hat{V}^{\pi}(s'; w^{\pi}) - \hat{V}^{\pi}(s; w^{\pi})\Big)\nabla_{w^{\pi}} \hat{V}^{\pi}(s; w^{\pi})$$

under an approximation scheme $\hat{V}^{\pi}$. Similarly, the SARSA update rule

$$Q^{\pi}(s,a) \leftarrow Q^{\pi}(s,a) + \alpha\Big(r + \gamma Q^{\pi}(s', \pi(s')) - Q^{\pi}(s,a)\Big)$$

for tabular representations becomes

$$w^{\pi} \leftarrow w^{\pi} + \alpha\Big(r + \gamma \hat{Q}^{\pi}(s', \pi(s'); w^{\pi}) - \hat{Q}^{\pi}(s,a; w^{\pi})\Big)\nabla_{w^{\pi}} \hat{Q}^{\pi}(s,a; w^{\pi})$$

under an approximation scheme $\hat{Q}^{\pi}$. Finally, the Q-learning update rule

$$Q^{*}(s,a) \leftarrow Q^{*}(s,a) + \alpha\Big(r + \gamma \max_{a' \in \mathcal{A}} \{Q^{*}(s', a')\} - Q^{*}(s,a)\Big)$$

for tabular representations under an approximation scheme $\hat{Q}^{*}$ becomes

$$w^{*} \leftarrow w^{*} + \alpha\Big(r + \gamma \max_{a' \in \mathcal{A}} \{\hat{Q}^{*}(s', a'; w^{*})\} - \hat{Q}^{*}(s,a; w^{*})\Big)\nabla_{w^{*}} \hat{Q}^{*}(s,a; w^{*}) \,.$$

These rules are applicable to any approximation architecture, linear or nonlinear. However, when using linear architectures they can be greatly simplified, because the gradient with respect to a parameter w_j is simply the corresponding basis function ϕ_j, for $j = 1, 2, \ldots, k$.

$$\text{TD: } w_j^{\pi} \leftarrow w_j^{\pi} + \alpha\Big(r + \gamma \phi(s')^{\top} w^{\pi} - \phi(s)^{\top} w^{\pi}\Big)\phi_j(s)$$

$$\text{SARSA: } w_j^{\pi} \leftarrow w_j^{\pi} + \alpha\Big(r + \gamma \phi(s', \pi(s'))^{\top} w^{\pi} - \phi(s,a)^{\top} w^{\pi}\Big)\phi_j(s,a)$$

$$Q\text{-learning: } w_j^{*} \leftarrow w_j^{*} + \alpha\Big(r + \gamma \max_{a' \in \mathcal{A}} \{\phi(s', a')^{\top} w^{*}\} - \phi(s,a)^{\top} w^{*}\Big)\phi_j(s,a)$$

More sophisticated learning approaches rely on retaining the desired value-function property in a batch manner by processing several samples collectively. A variety of least-squares techniques have been proposed for linear architectures giving rise to several least-squares reinforcement learning methods, such as least-squares temporal difference (LSTD) learning (Bradtke and Barto 1996), least-squares policy evaluation (LSPE) (Nedić and Bertsekas 2003), hybrid least-squares approximation (Johns et al. 2009), and least-squares policy iteration (LSPI) (Lagoudakis and Parr 2003). The parameters of a linear architecture can also be estimated using Linear Programming (de Farias and Van Roy 2003). Specialized learning algorithms have been proposed when using a kernel-based approximation architecture, based either on Gaussian process TD (GPTD) (Engel et al. 2003), Gaussian process SARSA (GPSARSA) (Engel et al. 2005), kernelized LSTD (KLSTD) and LSPI (KLSPI) (Xu et al. 2007), support vector regression (Bethke et al. 2008), or Gaussian process regression (Rasmussen and Kuss 2004; Bethke and How 2009). A unified view of kernelized value-function approximation is offered by Taylor and Parr (2009). On the other hand, boot-strapping – the updating of a value estimate based on other value estimates – is the main learning approach behind batch methods for nonlinear architectures, such as fitted Q-iteration (FQI) (Ernst et al. 2005).

Examples

Very close approximations of the state value function of optimal policies in two well-known problems are presented to illustrate the difficulty of value-function approximation. To obtain these close approximations, a fine discretization of the two-dimensional state space into a uniform grid of 250 × 250 was used for representation. The state-action value function Q was initially computed using approximate policy iteration (a sparse-matrix version of LSPI) with a set of indicator basis functions over the state grid and all actions and 500 (s, a, r, s') samples for each one of the 187,500 discrete cells (s, a), until convergence to a near-optimal policy; the state value function V was extracted from the Q values.

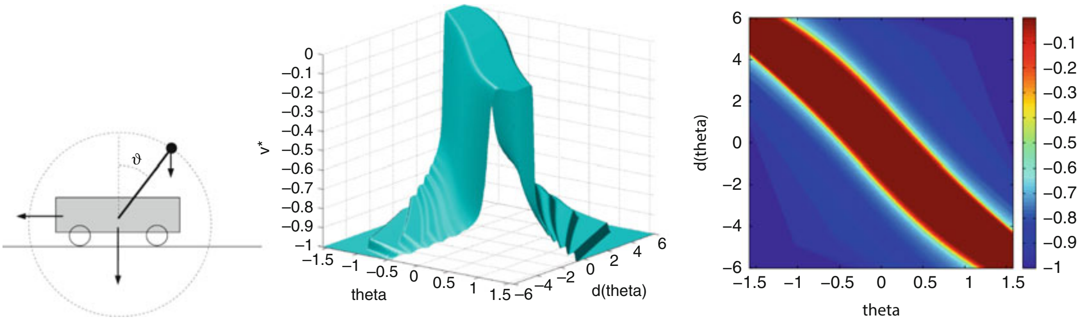
Inverted Pendulum

The *inverted pendulum* problem is to balance a pendulum of unknown length and mass at the upright position by applying forces to the cart it is attached to (Fig. 1, left). Three actions are allowed: left force LF (−50 Newtons), right force RF (+50 Newtons), or no force NF (0 Newtons). All three actions are noisy; Gaussian noise with $\mu = 0$ and $\sigma^2 = 10$ is added to the chosen action. The state space of the problem is continuous and consists of the vertical angle θ and the angular velocity $\dot{\theta}$ of the pendulum. The transitions are governed by the nonlinear dynamics of the system and depend on the current state and the current (noisy) control u:

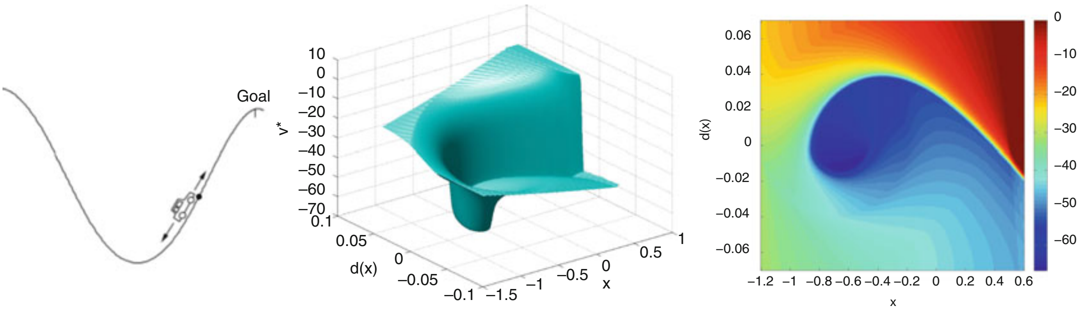
$$\ddot{\theta} = \frac{g\sin(\theta) - \alpha m l(\dot{\theta})^2\sin(2\theta)/2 - \alpha\cos(\theta)u}{4l/3 - \alpha m l\cos^2(\theta)},$$

where g is the gravity constant ($g = 9.8\,\mathrm{m/s^2}$), m is the mass of the pendulum (default: $m = 2.0\,\mathrm{kg}$), M is the mass of the cart (default: $M = 8.0\,\mathrm{kg}$), l is the length of the pendulum (default: $l = 0.5\,\mathrm{m}$), and $\alpha = 1/(m + M)$. The simulation step is $0.1\,\mathrm{s}$, thus the control input is given at a rate of $10\,\mathrm{Hz}$, at the beginning of each time step, and is kept constant during any time step. A reward of 0 is given as long as the angle of the pendulum does not exceed $\pi/2$ in absolute value (the pendulum is above the horizontal line). An angle greater than $\pi/2$ in absolute value signals the end of the episode and a reward (penalty) of -1. The discount factor of the process is 0.95.

Figure 1 shows a close approximation to the state value function V^* of an optimal policy for the inverted pendulum domain over the two-dimensional state space $(\theta, \dot{\theta})$. The value function indicates that states which potentially offer high return are clustered within a zone where θ and $\dot{\theta}$ have different signs and therefore the gravity force can be counteracted. Notice the nonlinearity of the function and the difficult approximation problem it presents.



Value Function Approximation, Fig. 1 Inverted pendulum: state value function of an optimal policy (3D and 2D) (Courtesy of Ioannis Rexakis)



Value Function Approximation, Fig. 2 Mountain car: state value function of an optimal policy (3D and 2D) (Courtesy of Ioannis Rexakis)

Mountain Car

The *mountain car* problem is to drive an underpowered car from the bottom of a valley between two mountains to the top of the mountain on the right (Fig. 2, left). The car is not powerful enough to climb any of the hills directly from the bottom of the valley even at full throttle; it must build some energy by climbing first to the left (moving away from the goal) and then to the right. Three actions are allowed: forward throttle FT (+1), reverse throttle RT (−1), or no throttle NT (0). All three actions are noisy; Gaussian noise with $\mu = 0$ and $\sigma^2 = 0.2$ is added to the chosen action. The state space of the problem is continuous and consists of the position x and the velocity $\dot{x}$ of the car along the horizontal axis. The transitions are governed by the nonlinear dynamics of the system and depend on the current state $(x(t), \dot{x}(t))$ and the current (noisy) control $u(t)$:

$$\begin{aligned} x(t+1) &= \text{B}{\small\text{OUND}}_x[x(t) + \dot{x}(t+1)] \\ \dot{x}(t+1) &= \text{B}{\small\text{OUND}}_{\dot{x}}[\dot{x}(t) \\ &\qquad + 0.001u(t) - 0.0025\cos(3x(t))] , \end{aligned}$$

where $\text{B}{\small\text{OUND}}_x$ is a function that keeps x within $[-1.2, 0.5]$, while $\text{B}{\small\text{OUND}}_{\dot{x}}$ keeps $\dot{x}$ within $[-0.07, 0.07]$. If the car hits the bounds of the position x, the velocity $\dot{x}$ is set to zero. A penalty of −1 is given at each step as long as the position of the car is below the right bound (0.5). As soon as the car position hits the right bound of the position, it has reached the goal; the episode ends successfully and a reward of 0 is given. The discount factor of the process is 0.99.

Figure 2 shows a close approximation to the state value function V^* of an optimal policy for the mountain car domain over the two-dimensional state space $(x, \dot{x})$. The value function indicates that in order to gain high return, the car has to follow a spiral in the state space that goes through states with progressively higher value. In practice, this means that the car has to move back and forth between the two mountains until sufficient energy is built to escape from the valley. Again, notice the high nonlinearity of the function and the hard approximation problem it presents.

Notation

The table summarizes the differences in names and symbols between the common notation (adopted here) and the alternative notation used in the literature.

Common notation		**Alternative notation**	
Name	**Symbol**	**Symbol**	**Name**
State space	$\mathcal{S}$	S	States
State	s, s'	i, j	State
Action space	$\mathcal{A}$	U	Controls
Action	a	u	Control
Transition model	$\mathcal{P}(s'\|s,a)$	$p_{ij}(u)$	Transition probabilities
Reward function	$\mathcal{R}$	g	Cost function
Discount factor	γ	α	Discount factor
Policy	π	μ	Policy
State value function	V	J	Cost-to-go function
State-action value function	Q	Q	Q-factors
Parameters/weights	w	r	Parameters
Learning rate	α	γ	Step size

Cross-References

- Curse of Dimensionality
- Dynamic Programming
- Feature Selection
- Gaussian Process Reinforcement Learning
- Least-Squares Reinforcement Learning Methods
- Q-Learning
- Radial Basis Function Approximation
- Reinforcement Learning
- Relational Value Iteration
- Temporal Difference Learning

Recommended Reading

Bertsekas DP, Tsitsiklis JN (1996) Neuro-dynamic programming. Athena Scientific, Belmont

Bethke B, How JP (2009) Approximate dynamic programming using Bellman residual elimination and Gaussian process regression. In: Proceedings of the American control conference, St. Louis, pp 745–750

Bethke B, How JP, Ozdaglar A (2008) Approximate dynamic programming using support vector regression. In: Proceedings of the IEEE conference on decision and control, Cancun, pp 745–750

Bradtke SJ, Barto AG (1996) Linear least-squares algorithms for temporal difference learning. Mach Learn 22(1–3):33–57

Buşoniu L, Babuška R, Schutter BD, Ernst D (2010) Reinforcement learning and dynamic programming using functions approximators. CRC, Boca Raton

de Farias DP, Van Roy B (2003) The linear programming approach to approximate dynamic programming. Oper Res 51(6):850–865

Engel Y, Mannor S, Meir R (2003) Bayes meets Bellman: the Gaussian process approach to temporal difference learning. In: Proceedings of the international conference on machine learning (ICML), Washington, DC, pp 154–161

Engel Y, Mannor S, Meir R (2005) Reinforcement learning with Gaussian processes. In: Proceedings of the international conference on machine learning (ICML), Bonn, pp 201–208

Ernst D, Geurts P, Wehenkel L (2005) Tree-based batch mode reinforcement learning. J Mach Learn Res 6:503–556

Johns J, Petrik M, Mahadevan S (2009) Hybrid least-squares algorithms for approximate policy evaluation. Mach Learn 76(2–3):243–256

Lagoudakis MG, Parr R (2003) Least-squares policy iteration. J Mach Learn Res 4:1107–1149

Mahadevan S, Maggioni M (2007) Proto-value functions: a Laplacian framework for learning representation and control in Markov decision processes. J Mach Learn Res 8:2169–2231

Menache I, Mannor S, Shimkin N (2005) Basis function adaptation in temporal difference reinforcement learning. Ann Oper Res 134(1):215–238

Nedić A, Bertsekas DP (2003) Least-squares policy evaluation algorithms with linear function approximation. Discret Event Dyn Syst Theory Appl 13(1–2):79–110

Parr R, Painter-Wakefield C, Li L, Littman M (2007) Analyzing feature generation for value-function approximation. In: Proceedings of the international conference on machine learning (ICML), Corvallis, pp 449–456

Puterman ML (1994) Markov decision processes: discrete stochastic dynamic programming. Wiley, New York

Rasmussen CE, Kuss M (2004) Gaussian processes in reinforcement learning. In: Thrun S, Saul LK, Scholkopf B (eds) Advances in neural information

processing systems (NIPS). MIT Press, Cambridge pp 751–759
Sutton R, Barto A (1998) Reinforcement learning: an introduction. MIT, Cambridge
Taylor G, Parr R (2009) Kernelized value function approximation for reinforcement learning. In: Proceedings of the international conference on machine learning (ICML), Toronto, pp 1017–1024
Xu X, Hu D, Lu X (2007) Kernel-based least-squares policy iteration for reinforcement learning. IEEE Trans Neural Netw 18(4):973–992

Variance Hint

▸ Inductive Bias

VC Dimension

Thomas Zeugmann
Hokkaido University, Sapporo, Japan

Motivation and Background

We define an important combinatorial parameter that measures the combinatorial complexity of a family of subsets taken from a given universe (learning domain) X. This parameter was originally defined by Vapnik and Chervonenkis (1971) and is thus commonly referred to as Vapnik-Chervonenkis dimension, commonly abbreviated as VC *dimension*. Subsequently, Dudley (1978, 1979) generalized Vapnik and Chervonenkis' (1971) results. The reader is also referred to Vapnik's (2000) book in which he greatly extends the original ideas. This results in a theory which is called structural risk minimization.

The importance of the VC dimension for ▸ PAC learning was discovered by Blumer et al. (1989) who introduced the notion to computational learning theory.

As Anthony and Biggs (1992, Page 71) have put it, "*The development of this notion is probably the most significant contribution that mathematics has made to Computational Learning Theory.*"

Recall that we use $|S|$ and $\wp(S)$ to denote the cardinality and the power set of any set S, respectively. We first define the VC dimension and provide a short explanation of its importance for ▸ PAC learning. Then we present some examples.

Definition

Let $X \neq \emptyset$ be any learning domain, let $\mathcal{C} \subseteq \wp(X)$ be any nonempty concept class, and let $S \subseteq X$ be any finite set. We set

$$\Pi_{\mathcal{C}}(S) = \{S \cap c \mid c \in \mathcal{C}\}\,.$$

1. S is said to be *shattered* by $\mathcal{C}$ iff $\Pi_{\mathcal{C}}(S) = \wp(S)$.
2. The VC *dimension* of $\mathcal{C}$ is the cardinality of the largest finite set $S \subseteq X$ that is shattered by $\mathcal{C}$.

   If arbitrary large finite sets S are shattered by $\mathcal{C}$, then the VC dimension of $\mathcal{C}$ is defined to be infinite.

Notation: By VC$(\mathcal{C})$ we denote the VC *dimension* of $\mathcal{C}$.

Remarks

As far as ▸ PAC learning is concerned, for a sample set S, the notion $\Pi_{\mathcal{C}}(S)$ has the following meaning. Essentially, $\Pi_{\mathcal{C}}(S)$ collects the set of *all subsets* of the sample set S which are made positive by some concept $c \in \mathcal{C}$. Consequently, $S \cap c$ represents the elements of S that are labeled as to be positive by the concept c. Hence, $\Pi_{\mathcal{C}}(S)$ is the collection of all such subsets taken over all $c \in \mathcal{C}$. If *every* subset of S can be labeled as to be positive by some concept $c \in \mathcal{C}$ and c does not make any other element of S positive, then S is shattered.

If VC$(\mathcal{C}) = d$ then there *exists* a finite set $S \subseteq X$ such that $|S| = d$, and S is shattered by $\mathcal{C}$. Moreover, *every* set $S \subseteq X$ with $|S| > d$ is *not* shattered by $\mathcal{C}$.

It is intuitively clear that an infinite VC dimension might enormously complicate learning. On the other hand, it is by no means obvious that a finite VC dimension may always guarantee the learnability of the corresponding concept class. However, this is a central theorem of the ▸ PAC learning theory. Moreover, the value of the VC dimension is a measure of the sample complexity. This holds for PAC learning and beyond. Further models where this is true comprise the ▸ online learning models (cf. Haussler et al. 1994; Maass and Turán 1990; Littlestone 1988), models of ▸ query-based learning (cf. Maass and Turán 1990), and others.

Examples

First, let $\mathcal{C}$ be any finite concept class. Then, since it requires 2^d distinct concepts to shatter a set of cardinality d, no set of cardinality larger than $\log|\mathcal{C}|$ can be shattered. Thus, $\log|\mathcal{C}|$ is always an upper bound for the VC dimension of finite concept classes. Here log denotes the logarithm to the base 2.

However, if the VC dimension can be determined, it usually gives a better bound than $\log|\mathcal{C}|$. To see this, let $\mathcal{L}_n = \{x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n\}$, $n \geq 1$ be a set of literals and let $X = \{0,1\}^n$ be the n-dimensional Boolean learning domain. Furthermore, let $\mathcal{C}_n \subseteq \wp(X)$ be the class of all concepts describable by a monomial including the empty monomial (representing $\{0,1\}^n$) and the conjunction of all literals (representing $\emptyset$). Then $|\mathcal{C}_n| = 3^n + 1$ and thus $\mathrm{VC}(\mathcal{C}) \leq n(\log 3) + 1$. But $\mathrm{VC}(\mathcal{C}_n) = n$ for all $n \geq 2$ and $\mathrm{VC}(\mathcal{C}_1) = 2$ as shown by Natschläger and Schmitt (1996). Note that the same is true for the class of all concepts describable by *monotone monomials*, i.e., monomials containing only non-negated literals.

Next, we consider the concept class $\mathcal{C}$ of all axis-parallel rectangles. So let $X = \mathbb{E}^2$ be the two-dimensional Euclidean space and $\mathcal{C} \subseteq \wp(\mathbb{E}^2)$ be the set of all axis-parallel rectangles, i.e., products of intervals on the x-axis with intervals on the y-axis. Then, it is not hard to see that $\mathrm{VC}(\mathcal{C}) = 4$.

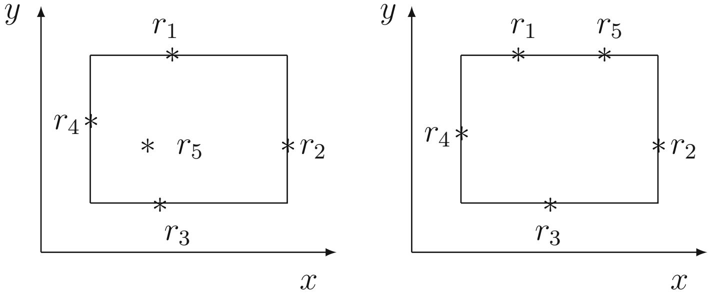
Clearly, we can shatter the empty set and sets of cardinality 1, 2, and 3. Now, let $S = \{r_1, r_2, r_3, r_4\}$ be such that r_1, r_2, r_3, r_4 are the middle points of the sides of some square. Then it is not hard to see that there are 16 concepts c_i, $1 \leq i \leq 16$, in $\mathcal{C}$ such that $\wp(S) = \{S \cap c_i \mid 1 \leq i \leq 16\}$. Hence, $\mathrm{VC}(\mathcal{C}) \geq 4$.

Next, let $S = \{r_1, r_2, r_3, r_4, r_5\}$ be any set of 5 pairwise different points. Let c be the smallest closed axis-parallel rectangle containing the points of S. Since c has only four sides, there must be some point $r \in S$, say r_5, such that r_5 lies either in the interior of c or r_5 lies on some side of c along with another point of S (cf. Fig. 1). Suppose S is shattered by $\mathcal{C}$. Then, there has to be a concept $c \in \mathcal{C}$ such that $\{r_1, r_2, r_3, r_4\} = S \cap c$. However, by construction we obtain that $\{r_1, r_2, r_3, r_4\} = S \cap c$ implies $r_5 \in S \cap c$, a contradiction. Thus, *no* set of cardinality 5 is shattered. Hence, $\mathrm{VC}(\mathcal{C}) = 4$.

The latter result can be easily generalized. Let $X = \mathbb{E}^n$, and let $\mathcal{C}$ be the set of all axis-parallel parallelepipeds in $\mathbb{E}^n$. Then $\mathrm{VC}(\mathcal{C}) = 2n$.

A further generalization is as follows. Let X be the real line (one-dimensional Euclidean

VC Dimension, Fig. 1 No set of cardinality 5 can be shattered by axis-parallel rectangles



space), i.e., $X = \mathbb{E}$, and let $\mathcal{C}$ be the set of all unions of at most s (closed or open) intervals for some fixed constant $s \geq 1$. Let $S = \{x_i \mid 1 \leq i \leq 2s,\ x_i < x_{i+1}$ for all $1 \leq i < 2s\}$. Then one easily verifies that S is shattered by $\mathcal{C}$. Hence, $\mathrm{VC}(\mathcal{C}) \geq 2s$. On the other hand, if S is any set of $2s + 1$ pairwise different points with $x_i < x_{i+1}$ for all $1 \leq i \leq 2s$, then no concept in $\mathcal{C}$ contains $x_1, x_3, \ldots, x_{2s+1}$ without also containing a point in $x_2, x_4, \ldots, x_{2s}$. Thus, no such S is shattered. Consequently, we have $\mathrm{VC}(\mathcal{C}) = 2s$.

Furthermore, we can generalize the observations made above by deriving some rules that turn out to be very useful to estimate the VC dimension of more complicated concept classes provided they can be constructed from simpler classes.

First, let $\mathcal{C}_1$ and $\mathcal{C}_2$ be concept classes such that $\mathcal{C}_1 \subseteq \mathcal{C}_2$. Then we clearly have

$$\mathrm{VC}(\mathcal{C}_1) \leq \mathrm{VC}(\mathcal{C}_2)\ .$$

Second, let X be any learning domain, let $\mathcal{C} \subseteq \wp(X)$ and define the complement of $\mathcal{C}$ to be $\overline{\mathcal{C}} = \{X \setminus c \mid c \in \mathcal{C}\}$. Then we have

$$\mathrm{VC}(\overline{\mathcal{C}}) = \mathrm{VC}(\mathcal{C})\ .$$

Third, consider two concept classes $\mathcal{C}_1$ and $\mathcal{C}_2$ defined over the same learning domain X. Let $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ be the union of $\mathcal{C}_1$ and $\mathcal{C}_2$. Then,

$$\mathrm{VC}(\mathcal{C}) \leq \mathrm{VC}(\mathcal{C}_1) + \mathrm{VC}(\mathcal{C}_2) + 1\ .$$

Fourth, let $\mathcal{C}$ be any concept class such that $\mathrm{VC}(\mathcal{C}) = d$. Consider the $\mathcal{C}_s$ union (or intersection) of at most s concepts from $\mathcal{C}$, where $s \geq 1$ is any fixed constant, i.e., $\mathcal{C}_s = \{c \mid c = \bigcup_{1 \leq i \leq s} c_i\ ,\ c_i \in \mathcal{C}\}$ (or $\mathcal{C}_s = \{c \mid c = \bigcap_{1 \leq i \leq s} c_i\ ,\ c_i \in \mathcal{C}\}$). Then one can show that

$$\mathrm{VC}(\mathcal{C}_s) \leq 2ds \cdot \log(3s)\ .$$

Numerous further examples can be found in, e.g., Vapnik and Chervonenkis (1974), Haussler and Welz (1987), Anthony and Bartlett (1999), Wenocur and Dudley (1981), Karpinski and Werther (1994), Karpinski and Macintyre (1995), Sakurai (1995), and Mitchell et al. (1999).

Applications

Let us return to the notion $\Pi_{\mathcal{C}}(S)$ and let us generalize it a bit as follows. For any natural number $m \in \mathbb{N}$ and any nonempty concept class $\mathcal{C} \subseteq \wp(S)$, we set

$$\Pi_{\mathcal{C}}(m) = \max\{|\Pi_{\mathcal{C}}(S)| \mid S \subseteq X,\ |S| = m\}\ .$$

We can use the new notion to give an equivalent definition of the VC dimension of a concept class $\mathcal{C}$, i.e.,

$$\mathrm{VC}(\mathcal{C}) = \max\{d \mid d \in \mathbb{N},\ \Pi_{\mathcal{C}}(d) = 2^d\}\ .$$

Looking at $\Pi_{\mathcal{C}}(m)$ from the perspective of learning, we see the following. The argument m refers to the sample size. $\Pi_{\mathcal{C}}(m)$ is describing the maximum number of ways a sample of size m can be labeled by concepts taken from $\mathcal{C}$. Hence, the number $\Pi_{\mathcal{C}}(m)$ behaves as a measure of concept class complexity. What can be said about $\Pi_{\mathcal{C}}(m)$? Suppose $d = VC(\mathcal{C})$; then $m \leq d$ implies $\Pi_{\mathcal{C}}(m) = 2^m$. On the other hand, $m > d$ directly implies $\Pi_{\mathcal{C}}(m) < 2^m$. Therefore, we are interested in learning how fast $\Pi_{\mathcal{C}}(m)$ really grows provided $m > d$. The key ingredient to obtain the desired information is usually referred to as Sauer's Lemma (cf. Sauer 1972). Under the assumptions made above, it states that

$$\Pi_{\mathcal{C}}(m) \leq \sum_{i=0}^{d} \binom{m}{i},$$

$$\text{where} \quad \binom{m}{i} = 0 \quad \text{if } i > m\ .$$

Like many important results, Sauer's Lemma (cf. Sauer 1972) has several proofs and generalizations have been studied, too. We refer the reader to Anthony and Biggs (1992), Kearns and Vazirani (1994), and Gurvits (1997) for a more detailed exposition.

Let us first look at the case $m \le d$ already considered. For this case Sauer's Lemma is telling us that

$$\Pi_{\mathcal{C}}(m) \le \sum_{i=0}^{d} \binom{m}{i} = 2^m,$$

and thus, we get an exponential bound. The interesting aspect is that in the remaining cases, the bound is *polynomial*. For simplifying notation, we set

$$\Phi(d,m) = \sum_{i=0}^{d} \binom{m}{i}.$$

Using combinatorial arguments and Stirling approximation, one can show that

1. $\Phi(0,m) = \binom{m}{0} = 1$ for all $m \in \mathbb{N}$.
2. $\Phi(d,1) = \binom{1}{0} + \binom{1}{1} = 2$ for all $d \in \mathbb{N}, d \ge 1$.
3. $\Phi(d,m) = \Phi(d,m-1) + \Phi(d-1,m-1)$ for all $d,m \in \mathbb{N}, d \ge 1,\ m \ge 2$.
4. $\Phi(d,m) \le m^d + 1$ for all $d \ge 0, m \ge 0$.
5. $\Phi(d,m) \le m^d$ for all $d \ge 2, m \ge 2$.
6. $\Phi(d,m) \le (\frac{em}{d})^d$ for all $m \ge d \ge 1$.

That is, (4) through (6) provide a bound polynomial in m for $\Pi_{\mathcal{C}}(m)$ whenever VC($\mathcal{C}$) is finite. This insight is fundamental for ▶ PAC learning and other learning models.

Linial et al. (1991) initiated the study of the complexity problem of computing the VC dimension of a finite family of concepts defined over a finite learning domain. Given any finite learning domain X of cardinality n and any concept class $\mathcal{C} \subseteq \wp(X)$ of cardinality r, one can represent the concept class $\mathcal{C}$ by an $r \times n$ matrix M such that $M_{ij} = 1$ iff $x_j \in c_i$. Then each row of M represents a concept $c \in \mathcal{C}$ and each column represents a point in X. The *discrete VC dimension decision problem* is then, given a $\{0,1\}$-valued matrix M and an integer $d \ge 1$ as input, to decide whether or not $VC(\mathcal{C}) \le d$, and the *discrete VC dimension problem* is, given a $\{0,1\}$-valued matrix M as input, to determine $VC(\mathcal{C})$.

Linial et al. (1991) showed that the discrete VC dimension decision problem to be solvable in time $O(rn^d)$ and that the discrete VC dimension problem can be solved in time $O(rn^{\log r})$. Further progress was made by Shinohara (1995) who showed that the discrete VC dimension decision problem is in the complexity class $\mathrm{SAT}_{\log^2 n}$ and hard for the complexity class $\mathrm{SAT}^{\mathrm{CNF}}_{\log^2 n}$, where $\mathcal{P} \subseteq \mathrm{SAT}^{\mathrm{CNF}}_{\log^2 n} \subseteq \mathrm{SAT}_{\log^2 n} \subseteq \mathcal{NP}$ (see Shinohara (1995) for details). Moreover, Papadimitriou and Yannakakis (1996) defined a new complexity class $\mathcal{LOGNP}$ and showed the VC dimension decision problem to be complete for this class.

However, the matrix representation of a concept class may be exponentially larger than a parameterized representation of it, e.g., the concept class may be generated by a circuit. Representing concept classes by circuits, Schaefer (1999) showed the discrete VC dimension problem (modified in the canonical way) to be Σ_3^p complete. For a definition of the complexity class Σ_3^p, we refer to Arora and Barak (2009).

Furthermore, we refer the reader to Goldberg and Jerrum (1995) who succeeded in bounding the VC dimension of concept classes parameterized by real numbers.

Finally, the notion of the VC dimension can be generalized to sets of indicator functions and to sets of real functions (cf. Vapnik 2000, Section 3.6). These generalizations play an important role in statistical learning theory.

Cross-References

▶ Epsilon Nets
▶ PAC Learning
▶ Statistical Machine Translation
▶ Structural Risk Minimization

Recommended Reading

Anthony M, Bartlett PL (1999) Neural network learning: theoretical foundations. Cambridge University Press, Cambridge

Anthony M, Biggs N (1992) Computational learning theory. Cambridge tracts in theoretical computer science, Vol 30. Cambridge University Press, Cambridge

Arora S, Barak B (2009) Computational complexity: A Modern approach. Cambridge University Press, Cambridge

Blumer A, Ehrenfeucht A, Haussler D, Warmuth MK (1989) Learnability and the Vapnik-Chervonenkis dimension. J ACM 36(4):929–965

Dudley RM (1978) Central limit theorems for empirical measures. Ann Probab 6(6):899–929

Dudley RM (1979) Corrections to "Central limit theorems for empirical measures". Ann Probab 7(5):909–911

Goldberg PW, Jerrum MR (1995) Bounding the Vapnik-Chervonenkis dimension of concept classes parameterized by real numbers. Mach Learn 18(2-3):131–148

Gurvits L (1997) Linear algebraic proofs of VC-dimension based inequalities. In: Ben-David S (ed) Proceedings of the third european conference on computational learning theory, EuroCOLT '97, Jerusalem, Israel, March 1997, Lecture notes in artificial Intelligence, vol 1208. Springer, pp 238–250

Haussler D, Littlestone N, Warmuth MK (1994) Predicting $\{0, 1\}$ functions on randomly drawn points. Info Comput 115(2):248–292

Haussler D, Welz E (1987) Epsilon nets and simplex range queries. Discret Comput Geom 2:127–151

Karpinski M, Macintyre A (1995) Polynomial bounds for VC dimension of sigmoidal neural networks. In: Proceedings of the 27th annual ACM symposium on theory of computing, ACM Press, New York, pp 200–208

Karpinski M, Werther T (1994) VC dimension and sampling complexity of learning sparse polynomials and rational functions. In: Hanson SJ, Drastal GA, Rivest RL (eds) Computational learning theory and natural learning systems. Constraints and prospects, vol I, chap. 11. MIT Press, pp 331–354

Kearns MJ, Vazirani UV (1994) An Introduction to computational learning theory. The MIT Press, Cambridge, Massachusetts

Linial N, Mansour Y, Rivest RL (1991) Results on learnability and the Vapnik-Chervonenkis dimension. Inform Comput 90(1):33–49

Littlestone N (1988) Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. Mach Learn 2(4):285–318

Maass W, Turán G (1990) On the complexity of learning from counterexamples and membership queries. In: Proceedings of the 31st annual symposium on foundations of computer science (FOCS 1990), St. Louis, 22-24 October 1990. IEEE Computer Society Press, Los Alamitos, pp 203–210

Mitchell A, Scheffer T, Sharma A, Stephan F (1999) The VC-dimension of subclasses of pattern languages. In: Watanabe O, Yokomori T (eds) Proceedings of the 10th international conference on algorithmic learning theory, ALT '99, Tokyo, Dec 1999, Lecture notes in artificial intelligence, vol 1720. Springer, pp 93–105.

Natschläger T, Schmitt M (1996) Exact VC-dimension of Boolean monomials. Infor Process Lett 59(1): 19–20

Papadimitriou CH, Yannakakis M (1996) On limited nondeterminism and the complexity of the V-C dimension. J Comput Syst Sci 53(2):161–170

Sakurai A (1995) On the VC-dimension of depth four threshold circuits and the complexity of Boolean-valued functions. Theoret Comput Sci 137(1):109–127

Sauer N (1972) On the density of families of sets. J Comb Theory (A) 13(1):145–147

Schaefer M (1999) Deciding the Vapnik-Červonenkis dimension is Σ_3^p-complete. J Comput Syst Sci 58(1): 177–182

Shinohara A (1995) Complexity of computing Vapnik-Chervonenkis dimension and some generalized dimensions. Theoret Comput Sci 137(1):129–144

Vapnik VN (2000) The nature of statistical learning theory, 2nd edn. Springer, Berlin

Vapnik VN, Chervonenkis AY (1971) On the uniform convergence of relative frequencies of events to their probabilities. Theory Probab Appl 16(2):264–280

Vapnik VN, Chervonenkis AY (1974) Theory of pattern recognition. Nauka, Moskwa (In Russian)

Wenocur RS, Dudley RM (1981) Some special Vapnik-Chervonenkis classes. Discret Math 33:313–318

Vector Optimization

▶ Multi-objective Optimization

Version Space

Claude Sammut
The University of New South Wales, Sydney, NSW, Australia

Definition

Mitchell (1977, 1982) defines the *version space* for a learning algorithm as the subset of hypotheses consistent with the training examples. That is, the ▶ hypothesis language is capable of describing a large, possibly infinite, number of concepts. When searching for the target concept, we are only interested in the subset of sentences in the hypothesis language that are consistent with the training examples, where consistent means that the examples are correctly classified (assuming deterministic concepts and no ▶ noise in the data). While the version space may be infinite, it can often be represented in a compact manner by maintaining only its *bounds*, the most specific (▶ Most Specific Hypothesis) and ▶ most general hypotheses. Any hypothesis that is more general than a hypothesis in the most specific bound and more specific than a hypothesis in the most general bound is in the version space.

Cross-References

▶ Learning as Search
▶ Noise

Recommended Reading

Mitchell TM (1977) Version spaces: a candidate elimination approach to rule-learning. In: Proceedings of the fifth international joint conference on artificial intelligence, Cambridge, pp 305–310

Mitchell TM (1982) Generalization as search. Artif Intell 18(2):203–226

Viterbi Algorithm

A dynamic programming algorithm for finding the most likely sequence of hidden states resulting in an observed sequence of output events. The most likely sequence is called the Viterbi path. The Viterbi algorithm was popularized due to its usability in Hidden Markov models (HMM).

The Viterbi algorithm was initially proposed by Andrew Viterbi as an error-correction scheme for noisy digital communication links. It is now also commonly used in speech recognition, natural language processing, and bioinformatics.

Recommended Reading

Viterbi AJ (1967) Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Trans Inf Theory 3(2):260–269

W

Web Advertising

▶ Text Mining for Advertising

Weight

Risto Miikkulainen
Department of Computer Science, The
University of Texas at Austin, Austin, TX, USA

Synonyms

Connection strength; Synaptic efficacy

Definition

In a ▶ neural networks, connections between neurons typically have weights that indicate how strong the connection is. The neuron computes by forming a weighted sum of its input, i.e., the activation of each input neuron is multiplied by the corresponding connection weight. Adapting such weights is the most important way of learning in neural networks. Connection weights are loosely modeled after the synaptic efficacies in biological neurons, where they determine how large a positive or negative change in the membrane potential each input spike generates (see ▶ Biological Learning: Synaptic Plasticity, Hebb Rule and Spike Timing Dependent Plasticity). In most models, all connection parameters are abstracted into a weight: attenuation or interaction of the potentials and connection delays are usually not taken into account. The weights are usually real-valued numbers ($-\infty \ldots \infty$), although in some algorithms, intended for VLSI implementation, the range and precision of these values can be restricted (or weights eliminated altogether). Weights in some methods can be restricted to positive values if the inputs are known to be positive and the method is based on comparing the similarity to the weights (as in e.g., ▶ Self-Organizing Maps, ▶ Adaptive Resonance Theory, and ▶ Radial Basis Function Networks). Most learning methods are based on adjusting the weight values. The weights are often initialized to small random values, although if enough is known about the input space and the task, more systematic initialization can improve performance significantly. The weights are then adjusted based on local information that is available on either side of the connection. Usually, only small modifications are made in each learning step to avoid disrupting what the network already knows, and learning converges over time to a setting of values that solves the task.

Within-Sample Evaluation

▶ In-Sample Evaluation

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

Word Sense Disambiguation

Rada Mihalcea
University of North Texas, Denton, TX, USA

Synonyms

Learning word senses; Solving semantic ambiguity

Definition

Ambiguity is inherent to human language. In particular, word sense ambiguity is prevalent in all natural languages, with a large number of the words in any given language carrying more than one meaning. For instance, the English noun *plant* can mean *green plant* or *factory*; similarly the French word *feuille* can mean *leaf* or *paper*. The correct sense of an ambiguous word can be selected based on the context where it occurs, and correspondingly the problem of word sense disambiguation is defined as the task of automatically assigning the most appropriate meaning to a polysemous word within a given context.

Motivation and Background

Word sense disambiguation is considered one of the most difficult problems in natural language processing, due to the high semantic ambiguity that is typically associated with language. It was first noted as a problem in the context of machine translation, when Warren Weaver, in his famous 1949 memorandum, pointed out word ambiguity as one of the problems that needed to be solved in order to enable automatic translations between the languages of the world (Weaver 1995). More than 50 years later, word sense ambiguity is still regarded as an important and difficult research problem, and it has been demonstrated to have a potentially significant impact on several natural language processing applications.

Applications

In addition to machine translation, the role of word sense disambiguation has also been explored in connection to other applications, such as monolingual information retrieval, cross-language information retrieval, question answering, knowledge acquisition, information extraction, text classification, and others. In particular, a significant amount of work has been carried out in areas related to information retrieval, where the resolution of word ambiguity has been shown to have an impact on both the precision of the system (by allowing for matches only between identical word meanings in the query and in the documents), as well as the recall of the system (by performing query expansion using synonyms of selected word meanings).

Brief History

Over the years, the field of word sense disambiguation has undergone steady improvements in both quality and scope, moving from the rule-based systems using hand crafted knowledge that were popular in the 1970s and 1980s, to the more advanced corpus-based methods used in the 1990s, and to the current hybrid systems that rely on a mix of knowledge-based and corpus-based resources, minimizing the need of sense annotated data and taking advantage of the Web. The shift from small-scale rule-based systems to large-scale data-driven methods has also implied an increase in coverage, with early systems typically addressing a handful of ambiguous words for which hand-coded rules were available, while many of the current systems have the ability to address all or almost all content words in unrestricted text.

Methods

Current word sense disambiguation systems are divided into three main categories:

Knowledge-based: These systems rely mainly on information drawn from lexical resources, such as dictionaries or thesauruses. The Lesk algorithm (Lesk 1986) is one of the most well-known knowledge-based word sense disambiguation methods. It decides the meaning of a word based on a measure of similarity among the definitions provided by a dictionary. For instance, for the phrase *pine cone*, the algorithm will select the meaning of *kind of evergreen tree* for *pine*, and *fruit of evergreen tree* for *cone*, as these are the definitions with the highest lexical overlap among all the possible definitions provided by a dictionary.

Unsupervised corpus-based: These approaches typically consist of algorithms for clustering word sense occurrences in a corpus, without making explicit reference to a sense inventory. The clustering can be performed in a monolingual environment, in which case different word occurrences are represented by features derived from their immediate context (Schutze 1998). Alternatively, a clustering of word senses can also be performed using cross-lingual evidence drawn from the translations observed in a parallel corpus (Ng et al. 2003). This line of work is often referred to as word sense discrimination, as the word meanings are not disambiguated against a sense inventory, but are discriminated against each other.

Supervised corpus-based: These methods are the focus of the current chapter, and they consist primarily of machine learning algorithms applied on large sense-annotated corpora. Supervised algorithms have been typically applied to one word at a time, although experiments have also been carried out for their application to all words in unrestricted text. While sense-annotated corpora have usually been constructed by hand, recent work has also explored various approaches for the automatic generation of such data, which has been used successfully in conjunction with machine learning algorithms.

Structure of the Learning System

Among the various knowledge-based and data-driven word sense disambiguation methods that have been proposed to date, supervised systems have been constantly observed as leading to the highest performance. In these systems, the sense disambiguation problem is formulated as a supervised learning task, where each sense-tagged occurrence of a particular word is transformed into a feature vector, which is then used in an automatic learning process.

Given a target word and a set of examples where this word occurs, each occurrence being annotated with the correct sense, a supervised system will attempt to learn how to automatically annotate occurrences of the given word in new, previously unseen, contexts. This process is accomplished in two steps. First, representative features are extracted from the context of the ambiguous word; this step is applied to the annotated examples (training) as well as the unlabeled examples (test). Second, a machine learning algorithm is applied on the feature vectors, and consequently the most likely sense is assigned to the test occurrences of the target word.

Features

Research in supervised word sense disambiguation has considered two main types of features to model occurrences of ambiguous words:

$W-1$	$W+1$	$P-1$	$P+1$	Growth	Flowering	Industrial	Staff	Sense
Flowering	Helps	Adj	Verb	Y	Y	N	N	Green plant
Industrial	Is	Adj	Verb	N	N	Y	Y	Factory

Contextual features, which are extracted from the immediate vicinity of the ambiguous word. These features usually consist of the words before and after the target word (a window size of 3–10 words is typical), their parts of speech, words in a syntactic dependency with the target word (e.g., the subject of the verb, the noun modified by an adjective), position in the sentence, and the like. For instance, the adjective *green* could be one of the contextual features extracted from the context *the green plant* for the ambiguous word *plant*.

Topical features, which are represented by the words most frequently co-occurring with a given meaning of the target word. These words are usually determined by counting the number of times each word occurs in the context of a word meaning, divided by the total number of occurrences in the context of the word regardless of its meaning. For instance, the *factory* meaning of *plant* could have topical features such as *industrial* and *work*, whereas the *green plant* meaning of *plant* might have features such as *animal* and *water*.

As an example of feature vector construction, consider the following two contexts provided for the ambiguous word *plant*:

The/det growth/noun of/prep a/det seedling/noun into/prep a/det flowering/adj **plant**/noun helps/verb children/noun investigate/verb the/det conditions/noun that/prep plants/noun need/verb for/prep growth/ noun.

The/det operations/noun staff/noun in/prep an/det industrial/adj **plant**/noun is/verb typically/adv measured/verb in/prep asset/noun utilization/noun.

The following two feature vectors are constructed:

Machine Learning

Provided a set of feature vectors representing different occurrences of an ambiguous target word, the goal of the machine learning system is to learn how to predict the most likely sense for a new occurrence. The word sense disambiguation literature describes experiments with a large number of machine learning algorithms, including decision lists (Yarowsky 2000), instance-based learning (Ng and Lee 1996), Naïve Bayes and decision trees (Pedersen 1998), support vector machines (Lee and Ng 2002), and others. A comparison of several machine learning algorithms for word sense disambiguation is provided in Lesk (1986) and Mooney (1996).

Generation of Sense-Tagged Corpora

One of the main drawbacks associated with the supervised methods for word sense disambiguation is the cost incurred in the process of building sense-tagged corpora. Despite their high performance, the applicability of these supervised systems is limited to those few words for which sense-tagged data is available, and their accuracy is strongly connected to the amount of labeled data available at hand.

Sense annotations have been typically carried out by humans, which resulted in several publicly available data sets, such as those made available during the Senseval evaluations (http://www.senseval.org). However, despite the effort that went into the construction of these data sets, their applicability is limited to a handful of approximately 100 ambiguous words.

To address the sense-tagged data bottleneck problem, different methods for automatic sense-tagged data annotation have been proposed in the past, with various degrees of success. One such method relies on monosemous relatives extracted from dictionaries, which can be used to identify ambiguity-free occurrences in large corpora (Leacock et al. 1998; Mihalcea 1999). Another method relies on automatically bootstrapped disambiguation patterns, which can be used to generate a large number of sense-tagged examples (Mihalcea 2002; Yarowsky 1995). The use of volunteer contributors to create sense-annotated corpora has also been explored in the Open Mind Word Expert system (Chklovski and Mihalcea 2002). Finally, in recent work, Wikipedia was identified as a rich source of word sense annotations, which can be used to build supervised word sense disambiguation systems (Mihalcea 2007).

Cross-References

▶ Semi-supervised Text Processing

Recommended Reading

Agirre E, Edmonds P (2006) Word sense disambiguation: algorithms and applications. Springer, Berlin. http://www.wsdbook.org

Chklovski T, Mihalcea R (2002) Building a sense tagged corpus with open mind word expert. In: Proceedings of ACL 2002 workshop on WSD, Philadelphia

Leacock C, Chodorow M, Miller GA (1998) Using corpus statistics and wordnet relations for sense identification. Comput Linguist 24(1):147–165

Lee YK, Ng HT (2002) An empirical evaluation of knowledge sources and learning algorithms for word sense disambiguation. In: Proceedings of EMNLP 2002, Philadelphia

Lesk M (1986) Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In: SIGDOC 1986, Toronto

Mihalcea R (1999) An automatic method for generating sense tagged corpora. In: Proceedings of AAAI 1999, Orlando

Mihalcea R (2002) Bootstrapping large sense tagged corpora. In: Proceedings of LREC 2002, Las Palmas

Mihalcea R (2007) Using wikipedia for automatic word sense disambiguation. In: Proceedings of NAACL 2007, Rochester

Mihalcea R, Pedersen T (2005) Advances in word sense disambiguation. Tutorial presented at IBERAMIA 2004, ACL 2005, AAAI 2005. http://www.d.umn.edu/~tpederse/WSDTutorial.html

Mooney R (1996) Comparative experiments on disambiguating word senses: an illustration of the role of bias in machine learning. In: Proceedings of EMNLP, Philadelphia

Ng HT, Lee HB (1996) Integrating multiple knowledge sources to disambiguate word sense: an examplar-based approach. In: Proceedings of ACL, Santa Cruz

Ng HT, Wang B, Chan YS (2003) Exploiting parallel texts for word sense disambiguation: an empirical study. In: Proceedings of ACL, Sapporo

Pedersen T (1998) Learning probabilistic models of word sense disambiguation. Ph.D. dissertation. Southern Methodist University

Schutze H (1998) Automatic word sense discrimination. Comput Linguist 24(1):97–123

Weaver W (1995) Translation. In: Locke WN, Booth AD (eds) Machine translation of languages: fourteen essays. MIT Press, Cambridge, MA

Yarowsky D (1995) Unsupervised word sense disambiguation rivaling supervised methods. In: Proceedings of ACL, Cambridge, MA

Yarowsky D (2000) Hierarchical decision lists for word sense disambiguation. Comput Hum 34(1–2): 179–186

Word Sense Discrimination

Word sense discrimination is sometimes used as a synonym for ▸ word sense disambiguation. Note, however, that these two terms refer to somewhat different problems, as word sense discrimination implies a distinction between different word meanings in a corpus (without reference to a sense inventory), whereas word sense disambiguation refers to a sense assignment using a given sense inventory.

Z

Zero-One Loss

Zero-one loss is a common ► loss function used with ► classification learning. It assigns 0 to loss for a correct classification and 1 for an incorrect classification.

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1